Conception, Implementation and Empirical Evaluation of a Domain-Specific Language for Multi-Agent Traffic and Transport Simulations

by

BENJAMIN MANFRED HOFFMANN

A thesis submitted in partial fulfilment of the requirements of Edinburgh Napier University, for the award of

Doctor of Philosophy

School of Computing

Edinburgh Napier University

April 2023

Dedication

To Waltraud Winkels, Christoph Roth, and Claudia Sauer.

Author's declaration

I hereby declare that I, Benjamin Manfred Hoffmann, am the sole author and composer of this thesis. Furthermore, I declare that all sources used in researching and authoring this work are fully acknowledged by proper identification of quotations and by provision of detailed references of said sources.

I also declare that this thesis has not been submitted for any other degree or professional qualification.

BENJAMIN MANFRED HOFFMANN, 12TH OF APRIL, 2023

Abstract

Conception and implementation of agent-based simulation programs is a complex task. One of the key problems is the requirement of technical agent-based software engineering expertise on the one hand and professional knowledge of the application domain on the other. Either skill set is rare and only few people possess adequate knowledge of both domains. A joint development of software engineers and domain experts is often impeded by inaccurate communication resulting from the incompatible terminologies of the technical and the application domain. This is especially problematic since every change commences a new development cycle based on inapt communication. Domain-specific languages (DSLs) are a promising approach to overcome this gap. A well-designed concrete syntax can serve as the communicational basis. An expressive meta-model in combination with a concise concrete syntax allows modelling on a more abstract level close to the application domain. Via pre-defined transformations, executable simulation software can be generated from DSL-models. DSLs thus have the potential to increase the quality of the software and at the same time accelerate the entire development process. Realisation of this potential demands a perfectly designed language which in turn renders a proper language evaluation indispensable. However, this crucial step is often neglected by DSL developers. Therefore, there is a general demand for further contributions in this area of language engineering. This thesis presents the development of a DSL for the domain of agent-based traffic simulation and vehicle-routing optimisation together with a comprehensive empirical language evaluation. It depicts how an expressive meta-model was developed and merged with a concise concrete textual syntax. It also presents transformations that allow the generation of executable software for different platforms. Most importantly, it provides empirical evidence that language users with little programming knowledge as well as modellers with advanced software development skills can benefit from the application of the DSL in terms of software quality and development time.

Table of Contents

List of Tables

Tables **Page**

List of Figures

List of Acronyms

ABM agent-based modelling

ABMS agent-based modelling and simulation

ATAP accelerating test automation platform

ABS agent-based simulation

ACS ant-colony system

AOSE agent-oriented software engineering

API application programming interface

ASL adaptive storyline language

AST abstract syntax tree

ATL adaptive topic language

B2B business-to-business

B2C business-to-consumer

BCRC best cost route crossover

BDI belief-desire-intention

CAGR compound annual growth rate

CAS complex adaptive system

COM component object model

CIM computationally independent model

CDNF cognitive dimensions of notations framework

CVRP capacitated vehicle routing problem

DES discrete event simulation

EBNF extended Backus-Naur form

EMF Eclipse modeling framework

DMP data management plan

DSL domain-specific language

DSS decision support system

DSML domain-specific modelling language

EV electric vehicle

FQAD framework for qualitative assessment of DSLs

GIS geographic information systems

GPL general-purpose language

GUI graphical user interface

HTML hyper text markup language

IDE integrated development environment

ILP integer linear program

IS information systems

ITS intelligent transportation systems

LML last mile logistics

LOC line of code

MAS multi-agent system

MDSD model-driven software development

MOE measure of effectiveness

MTSP multiple travelling salesman problem

RMTSP restricted multiple travelling salesman problem

M2M model-to-model

M2T model-to-text

MTZ Miller-Tucker-Zemlin

OCL object constraints language

ODM origin-destination matrix

PDT Prometheus design tool

PIM platform independent model

PML pedagogical modelling language

PPM points per minute

PRT pedagogical relationship type

PSM platform-specific model

PUR pedagogical update rule

PHEM passenger car and heavy-duty emission model

RCS rate correct score

SAR storyline adaptation rule

SD system dynamics

SLE software language engineer

SLR systematic literature review

SMS systematic mapping study

TAR topic adaptation rule

THM Technische Hochschule Mittelhessen

TSP travelling salesman problem

UML unified modelling language

UI user interface

V2X vehicle-to-everything

VAP vehicle-actuated programming

VLE virtual learning environment

VR virtual reality

VRP vehicle routing problem

VRPTW vehicle routing problem with time windows

ZEZ zero emission zone

Listings

First Chapter

Introduction

1.1 Challenges of last mile logistics

Last mile logistics (LML) has drawn considerable interest of researchers from various fields. Though there is no universally agreed definition for the term (Boysen *et al.*, 2021), LML is widely understood to refer to the final sequence in a delivery process (Lim *et al.*, 2018). The great interest in this topic is easily understood by looking at the global impact of LML – both from an *economic* and from and *ecologic* point of view: in 2020, the value of the world-wide market for LML amounted to USD 18.7 Bn and forecasts predict a further growth to USD 62.7 Bn by 2027 (All the Research, 2021).

The already continuously rising demand for last mile deliveries received an additional impetus by the outbreak of the SARS-CoV 19 pandemic in 2020 (WEF, 2021). The imposition of lockdowns as a means to curb the spread of the virus also led to restricted access to brick and mortar shops and thus coerced consumers from large parts of the world into shopping online (Quak and Kin, 2020). As revenues grow, so do the demands of customers who do not only expect ever increasing delivery speeds but also more influence on the parameters of their delivery (WEF, 2020; Quak and Kin, 2020). A concrete example in this regard are delivery services offered by supermarkets. While these services are becoming increasingly popular, they have to meet time windows that leave less and less leeway for drivers to execute deliveries.

In a survey that asked 300 senior logistics and fulfilment executives for challenges in last mile delivery, increasing costs and a reliable fulfilment of customers' orders were the two most frequently given answers (Statista, 2021). High costs are a special problem of the last mile segment as it is usually regarded as the most ineffective part of the delivery chain (Soti, 2020). However, passing on delivery costs may make matters even worse: in a survey conducted among 2,000 Americans in 2019, 63 % of the respondents stated too high charges for delivery as a primary reason for them to refrain from checking out their shopping cart (ForterTeam, 2019).

Another serious problem related to last mile deliveries is their substantially negative impact on the environment. According to a report published by the World Economic Forum, 'delivery vehicles – both trucks and vans – add disproportionately high amounts [of urban CO_2 emissions] compared to passenger cars' (WEF, 2020, p. 7). In the 100 biggest cities[1], last mile deliveries emitted 19 Million tonnes of CO_2 in 2019 (WEF, 2020). Without any changes to the underlying delivery infrastructure, the number of delivery vehicles in these cities is predicted to rise from 5.3 Million in 2019 to 7.2 Million in 2030, causing and additional six million tonnes of CO_2 emissions and prolonging an average commuter trip from 53 minutes in 2019 to 64 minutes in 2030 (WEF, 2020).

To put it succinctly, practitioners in the field of LML are pressured to find and apply the means to deliver an increasing amount of goods within ever tightening time windows in a cost-effective and environmentally acceptable way while facing the problem of increasing congestions in urban areas. Fortunately, there exist several possible approaches towards these imminent challenges. Switching from conventional delivery vehicles that rely on combustion engines to electric vans and trucks is one approach to reduce CO_2 emissions. While electric vehicles still contribute to increased congestion, airborne delivery drones have the potential to be environmentally friendly without causing additional road traffic.

[1]Note: the report speaks of 'top 100 cities' without further specification.

Shifting the paradigm of how deliveries on the last mile are performed is another way that promises less CO_2 emissions, reduced congestion and even considerable cost reductions: Stationary pick-up points, lockers and micro-hubs have exhibited promising results and are expected to be key interventions in the future of last mile deliveries (Ballare and Lin, 2020; Quak and Kin, 2020). Another approach to avoid unnecessary emissions is the calculation of optimal delivery routes for all vehicles under consideration of aspects such as the current traffic situation (delivery vehicles stuck in traffic jams only exacerbate the situation), labour times (parcels not delivered due to the end of the driver's shift cause additional emissions on the next tour), or stipulated time windows (failed deliveries have to be returned and performed on the next day). Unfortunately, lacking the ability to calculate adequate delivery routes is among the major challenges the field of LML currently faces (Skylar Ross, March 2021).

1.2 The need for simulation in last mile logistics

Even though technical and organisational interventions with the potential to alleviate the aforementioned problems exist and are continuously refined and improved, practitioners still need a way to analyse and evaluate the results that can be expected from applying these interventions in different contexts (each intervention on its own or, perhaps even more interestingly, in combination with other approaches). However, an approach in which interventions of interest are implemented and observed in the field is often impractical due to a combination of imponderable risks, unreasonably high monetary or temporal investments. In such situations, where a system's reaction to a treatment is of focal interest, but this reaction cannot be directly observed because the application of the treatment is too dangerous, too

costly or not possible (with reasonable effort) for other reasons, simulation is the approach of choice (Shannon, 1998). Shannon defines *simulation* as

> '[...] the process of designing a model of a real system and conducting experiments with this model for the purpose of understanding the behaviour of the system and/or evaluating various strategies for the operations of the system' (Shannon, 1998, p. 7).

What is especially important about this definition is that it puts emphasis on the fact that simulation comprises two important aspects: in the first step, an appropriate model of the system of interest must be created, i.e. it must be decided, which aspects of the system are relevant and which are not. The relevant attributes together with their mutual relations will then find their way into the model. In the second part, the model is then *executed* or *applied* to investigate the behaviour of the system as it either evolves through the interactions of its constituting elements or as it reacts (and also evolves) to a modification or external stimulus.

In recent decades, computer simulations have become a crucial and widespread technique throughout many academic disciplines. Axelrod (1997) places simulation next to *induction* and *deduction* as a 'third way of doing science' (Axelrod, 1997, p. 5). The potential of computer simulations in the field of traffic and transport analysis and optimisation has long since been recognised. Clark and Daigle (1997), for example, discuss and exemplify how computer simulations support the evaluation of alternative traffic design and control strategies in the field of intelligent transportation systems (ITS). In their paper, the authors also provide a demonstration of how computer simulations can be used as a sandbox environment that allows for an innocuous experimental application of newly developed traffic management algorithms and/or systems.

Though the presented simulation applications are shown to be valuable supporting tools, their underlying software technologies, platforms and paradigms considerably deteriorated in the course of recent years. The tools suffer from considerable limitations in terms of maintainability and extensibility – especially in

light of the advent of more sophisticated modelling and simulation paradigms which also promise further advantages with regard to their explanatory power. Similar to the field of software development, the field of complex system simulation has seen continuous development which brought about several different simulation paradigms and technologies.

1.3 The Agent-based modelling approach

System dynamics (SD), discrete event simulation (DES), and agent-based modelling and simulation (ABMS) are widely considered the three prevalent modelling and simulation paradigms (Borshchev and Filippov, 2004; Maidstone, 2012). SD and DES are generally viewed to be rather conventional or traditional approaches compared to ABMS which is a comparatively new approach (Borshchev and Filippov, 2004; Chan *et al.*, 2010). What these three paradigms have in common is that they offer means to represent both the present state of a system in terms of its constituting entities (and their respective properties) and also a set of rules and principles that apply to the entities and that determine the future state of the system (Borshchev and Filippov, 2004). Each of the three approaches offers a unique set of benefits but also comes with a certain number of deficiencies. The suitability of each of the three paradigms solely depends on the system to be modelled (Borshchev and Filippov, 2004).

Of the three modelling methodologies, SD is the one that takes the most aggregated (or macro level) approach in order to represent and simulate a complex system (Maidstone, 2012; Borshchev and Filippov, 2004). In SD, *feedback loops* which can be either positive or negative illustrate the causal relations between variables; these feedback loops are complemented by *flows* between *stocks* of objects. Flows between stocks may be subject to *delays* which alter the dynamics of the system (Sterman, 2000). An important aspect of SD models is their *deterministic* behaviour, i.e. multiple simulation runs will yield the exact same result, so that multiple simulation runs are not necessary (Maidstone, 2012). SD models can be very efficiently applied

for problems with a very high abstraction level, e.g. population dynamics, ecosystems or macro traffic models; they are less suitable for problems that require a greater level of detail (Borshchev and Filippov, 2004).

In DES a system is viewed as a network in which entities flow between queues and servers which respectively collect and process these entities (Maidstone, 2012; Borshchev and Filippov, 2004). Though there are several different 'world-views' like event-scheduling or state-machines associated with DES (Chan *et al.*, 2010), a common characteristic is that the progress of a DES simulation is driven by the occurrence of various events like a new entity entering the simulation or the completion of some process on an entity within a server. In general, entities modelled in DES do not exhibit autonomous or proactive behaviour (Chan *et al.*, 2010), they are rather 'passive' (Maidstone, 2012) or 'reactive' (Chan *et al.*, 2010).

ABMS is an appropriate approach to get insight into systems in which patterns emerge from the interaction of several entities that exhibit complex and proactive behaviour (Bandini *et al.*, 2009; Chan *et al.*, 2010; Borenstein, 2015). In ABMS these entities are referred to as *agents*. Agents are capable of interacting with both their environment and other agents in an attempt to bring about a state of the system they regard as desirable or advantageous. Agent-based modelling (ABM) is applied in a great number of research fields, e.g. stock and consumer market simulations (Palmer *et al.*, 1999; North *et al.*, 2010; Farrenkopf *et al.*, 2014), investigation of predator-prey patterns (Mock *et al.*, 2007), or analysis of behavioural patterns in emergency situations (Carley *et al.*, 2006; Pan *et al.*, 2007).

This widespread adoption of ABMS in different academic fields might be the main reason why there is neither a broad agreement on an exact definition of ABMS, nor a universally accepted list of features that every agent must possess (Lind, 2001; Macal and North, 2010; Borshchev and Filippov, 2004; Bandini *et al.*, 2009; Chan *et al.*, 2010). According to the works of Lind (2001) and Wooldridge and Jennings (1995), there are three *notions of agency* that differ in the requirements that an agent must meet: the *very weak notion* does not impose any requirement and simply leaves it to the modellers and stakeholders to decide what is to be considered an agent; the *weak*

notion (Wooldridge and Jennings, 1995) requires an agent to exhibit *autonomous* (no human intervention is required), *social* (the agent interacts with other agents), *reactive* (the agent adapts its behaviour upon changes in its environment) and *pro-active* (the agent takes the initiative in order to achieve its goals) behaviour; the *stronger notion* (based on works of Yoav Shoham (1993) and Dennett (1987)) extends the weak notion in that it requires an agent to own features that correspond to cognitive properties found in human beings. The belief-desire-intention (BDI) paradigm which is rooted in the works of Rao and Georgeff (1991) and Bratman (1987) is an example that is based on agents that comply to the stronger notion of agency.

Macal and North (2010) define agent-based models as a composition of three essential components:

1. A set of agents with their respective features.
2. A topological definition on how the agents are related.
3. An environment within which the agents exist and perform their various activities.

Within the scope of this thesis, the term 'agent' is used in assumption of the weak notion of agency. The term ABMS is used for all simulation approaches that leverage an agent-based model. In doing so, this thesis follows Macal and North (2010) who note that agent-based simulation (ABS) is generally considered to focus on modelling dynamic processes while ABM is the more general term that also covers modelling for the purpose of optimisation. In this thesis, ABM is understood to refer to modelling approaches centred around the agent concept in order to represent a complex system. ABS, on the other hand, is understood as the proverbial second side of the same medal, i.e. to subsume all approaches that use an agent-based model in order to gain insight into further development of the modelled system.

In ABMS, no central authority dictates the behaviour of the agents in the system (Borshchev and Filippov, 2004; Macal and North, 2010). Rather, the system is

supposed to result from the individual behaviour of every single agent. Borshchev and Filippov (2004) note that this way ABMS allows modelling the entire system *from bottom to top* so that modellers do not need to possess an a-priori knowledge of the system behaviour at an aggregated level. Borshchev and Filippov (2004) and Macal and North (2010) recommend the application of ABMS for the modelling of systems in which autonomous entities interact, evolve and *actively* exhibit behaviour that may lead to new and/or unexpected *emergent phenomena* at the system level. Systems with these properties are referred to as complex adaptive systems (CASs).

1.4 Traffic networks as complex adaptive systems

Holland (1992) defines CASs as systems that are formed by a multitude of interdependent parts which act and interact according to an individual set of rules. These 'rule-based components' are not only capable of adapting their internal rules based on *credit assignment* and *rule discovery* but they also possess the ability of *anticipating* changes in the system. As Holland (1992) explains, in CAS the aggregated system behaviour cannot be predicted from merely looking at any single component in isolation. Instead, the system behaviour *emerges* as a result from a complex interplay of all components that constitute the system (Holland, 1992). For this reason, top-down approaches (like SD) or approaches in which the entities of the system are supposed to remain rather passive (as in DES) are less appropriate for modelling and simulation of CAS (Macal and North, 2010).

The structures and processes found in urban traffic (or traffic in general) suggest that trying to solve (real-world) traffic and transportation related problems is tantamount to the aspiration of a deeper insight into CASs: in real-world urban traffic scenarios vast numbers of participants follow their individual set of rules in an attempt to reach a set of individual goals. Against this background, it is not surprising that ABMS has attracted considerable interest within research approaches towards traffic and transport optimisation (e.g. Davidsson *et al.*, 2005; da Silva *et al.*, 2006; Bazzan and Klügl, 2014; Bazzan *et al.*, 2015).

1.5 Problems to address

Development of simulation based decision support system (DSS) that assist policy makers in the field of traffic and transportation management and optimisation is a challenging task (Davidsson *et al.*, 2005) that requires skills and expertise from two distinct bodies of knowledge (c.f., e.g. Borenstein, 2015): Not only does it require considerable knowledge and experience from the field of software development, but in order for the DSS to be of real value to its users, it is key that expert knowledge is sensibly incorporated into the system. However, only in the rarest of cases do software developers also happen to be experts in the field of traffic and transportation management. As a consequence, it is indispensable to the success of the development process that experts from both domains cooperate. However, in practice, failure of cooperation and communication between software and domain experts is one of the major reasons for unsuccessful software projects (Ghosh, 2011a; Dalal and Chhillar, 2012). Inconsistent terminologies used by the different stakeholders is one of the major problems that underlie the defective communication processes (Lu and Jin, 2000; Ghosh, 2011a). In addition to that, there are often diverging assumptions concerning the allocation of competences (Segal, 2009).

The application of ABMS is a first step towards an improved mutual understanding of software and domain experts: Macal (2016) points out that even without prior expertise, most people possess an intuitive understanding of the general nature and benefits of ABMS. The reason for this, according to Macal, is that the main task of an agent-based modeller is very similar to the way we form our decisions in our everyday life: imagining a given situation and then devising on possible courses of action and assessing their respective outcomes including the reactions they might invoke within our environment. However, even though ABMS is naturally related to human actions, reactions and interactions, the development of an agent-based application remains a highly challenging task with various intricacies (Parry, 2009; Vendrov *et al.*, 2014; Challenger *et al.*, 2016).

Vendrov *et al.* (2014) point to the languages used for the specification of agent-based models as a major source for the complexity of ABMS development: these mostly imperative languages require developers and domain experts to think at a very low level of abstraction; perhaps even worse, these languages mingle technical implementational details with domain processes in a way that obfuscates the intended domain-specific logic.. Obviously, such models do not provide a basis for an improved communication between developers and domain experts. The fact that the level of detail required in ABMS approaches is consistently increasing (Parry, 2009) and that the analysed domains are becoming ever more complex (Bazzan and Klügl, 2014) are catalysts for failing communication between developers and domain-experts which will inevitably lead to unsatisfactory results. For this reason, it is of crucial importance that domain experts thoroughly understand the models created by developers in order to provide informed and helpful feedback to developers (Ghosh, 2011a). Ideally, domain experts should even be able to create or at least modify such agent-based models on their own or in cooperation with a model developer.

The crucial importance of domain experts' ability to experiment with the models was already emphasized in a 1992 paper by Holland (1992). In Holland's vision, domain-experts with little to no expertise in computer science were able to modify system models without the need to deal with the implementational code. As this vision has not yet been sufficiently realised by current ABMS frameworks, this is a pressing issue that needs to be addressed in an appropriate way. North and Macal (2009) discuss the implementation of appropriate interfaces which enable domain experts to create and modify ABMS models as one of the key challenges in the field. Macal (2016) also lists four key challenges in the field of ABMSs: first, the need to increase the faith that stakeholders have in the models which requires less complex and more concise models; second, the necessity to enhance the transparency of models which is often negatively affected by the high complexity of current models; third, the acquisition of a broader understanding of how to efficiently build and

apply ABMS to obtain insightful data; and fourth, the need for an increase in the ease-of-use of tools designed for modellers so that the number of scientists that adopt an ABMS approach continues to grow.

1.6 Application of models to cope with high complexity

The field of model-driven software development (MDSD) offers powerful and auspicious approaches to overcome this dilemma (Challenger *et al.*, 2016). The MDSD methodology places models as the driving artefacts that directly determine the outcome of the development process (Brambilla *et al.*, 2012). For this, every MDSD approach requires the elicitation of relevant *concepts*, definition of appropriate *notations*, *processes and rules* as well the development of suitable *tools* (Brambilla *et al.*, 2012). Some MDSD approaches[2] are centred around special languages known as domain-specific languages (DSLs).

DSLs are languages which feature notations and abstractions that are tailored and restricted towards a particular domain of application in order to offer increased expressive power, enhanced ease of use and higher user productivity (van Deursen *et al.*, 2000; Mernik *et al.*, 2005; Barišic *et al.*, 2011; Barisic *et al.*, 2012b). The domain that a particular DSL is designed for is referred to as the *problem domain* or also *real-world domain* (Barisic *et al.*, 2012b). The set of technical tools, platforms and languages to represent and explore the problem domain is referred to as the *solution domain* (Barisic *et al.*, 2012a) (or *computation domain* (Barisic *et al.*, 2012b)). While general-purpose languages (GPLs) (e.g. Java or C++) are (in most cases) close to the computation domain (Barisic *et al.*, 2012b), DSLs can be considered as a bridge between the problem and the solution domain (Barišic *et al.*, 2011; Barisic *et al.*, 2012a; do Nascimento *et al.*, 2012). Based on this idea, Barisic *et al.* (2012b) define

[2] Völter and Benz (2013) refer to the development and application of domain-specific languages (DSLs) as a 'flavor' of MDSD (p.32).

DSLs as human-computer user interfaces (UIs) that aim to allow their users to think and express themselves in terms of ontological concepts taken from the problem domain.

In addition to the aforementioned benefits, DSLs are associated with a variety of positive impacts on software development projects: they are widely believed to improve communication between software and domain experts (Ghosh, 2011b), reduce the required programming skills (Barisic *et al.*, 2012b; Barisic *et al.*, 2012a), facilitate the understanding of programs/models (do Nascimento *et al.*, 2012; Poltronieri Rodrigues *et al.*, 2017), increase the level of abstraction (Iung *et al.*, 2020), reduce program/model complexity (do Nascimento *et al.*, 2012), improve the quality of the final product (Hermans *et al.*, 2009; Poltronieri Rodrigues *et al.*, 2017), and facilitate maintenance of source models (Barisic *et al.*, 2012a; Barisic *et al.*, 2012b).

Every DSL (and every computer language in general) comprises the following key components (c.f., e.g. Vangheluwe *et al.*, 2007; Hahn, 2008; Cuadrado *et al.*, 2013; Völter and Benz, 2013): an *abstract syntax* that defines a structure of *ontological concepts* (Atkinson and Kuhne, 2003) found in the problem domain. In other words, the abstract syntax defines which elements are supported by the language and how these are related to each other in the models[3] written in the DSL. As the abstract syntax defines the very 'essence' of its associated language (Vangheluwe *et al.*, 2007), it does not make any specifications with regard to the notation of the supported elements. The definition of concrete notations used to instantiate the abstract ontological concepts (in other words to create an actual model) is referred to as the *concrete syntax*. While the abstract syntax already defines some rules concerning the structure of the language elements, these rules are often not strict enough so that they would allow for models that violate certain laws of the domain. To render

[3]From this point on, the term 'model' will be used for a program or model specified with a DSL, whereas the term 'program' will be used for a program written in a GPL.

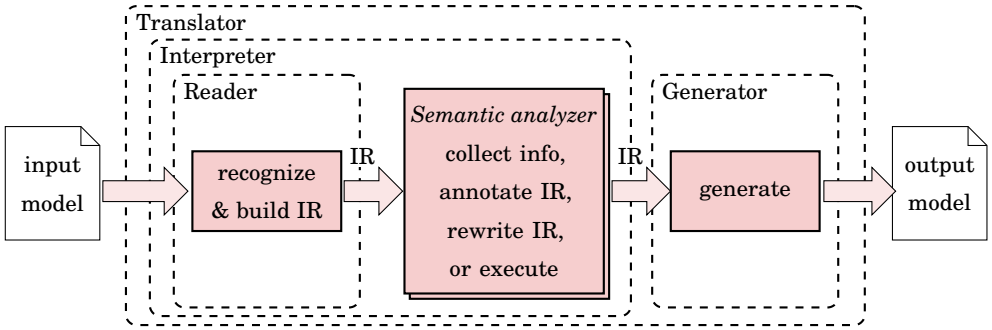


Figure 1.1: Components of a computer language in the language application process adapted from (Parr, 2011, p. 4).

these models invalid, most languages feature an additional set of *constraints* that must hold true within a model. These constraints (together with the rules from the abstract syntax) form the *static semantics* of the language.

The components mentioned so far allow the definition of models and a decision on whether the defined models are valid or not. What is still missing is a translation of models into computational activity. This computational activity that is expected upon the execution of models defines the *execution semantics* (Völter and Benz, 2013, p. 82) (also *dynamic semantics*) of a language. Consequently, the definition of a language's execution semantics requires the definition of a mechanism that maps every valid language model to a specific computational behaviour. In the context of this thesis, this mechanism will be provided by a *generator*. This generator comprises a set of *transformations* that translate a valid model written in the DSL into a valid model of a *target platform* that comprises one or several *target languages* each with a properly defined execution semantics. The execution semantics of the target languages then may again be defined by means of a set of transformations or they may directly be *interpreted* in order to produce the desired computational activity.

Figure 1.1 illustrates the interplay of all components in the language application process: an input model written in the concrete syntax of the language is applied to a *reader* (which often comprises a lexer and a parser (c.f., e.g. Parr, 2011)) that seeks to recognize the structure defined by the abstract syntax of the language. At this stage, a failing attempt to recognise the expected structure would result in a

compilation error. The reader builds an internal representation of the input model which is an instance of the abstract syntax of the language, known as an abstract syntax tree (AST). The AST is further processed by a *semantic analyser*. The actual steps taken by the semantic analyser vary according to the language. It might just cause the computer to perform various computational actions and thus *interpret* the internal representation. It can also act as a *validator* by applying the constraints of the static semantics. If the model is found to be valid, the internal representation is passed on to the generator which applies its transformation in order to create an *output model*. As was already mentioned, this output model might then be used as an input model in another iteration of the depicted process.

A common distinction between DSLs is the distinction between *textual* and *graphical* DSLs. Textual DSLs have a concrete syntax that is based on the use of characters whereas the concrete syntax of a graphical DSL is defined in terms of two-dimensional geometrical structures like rectangles, diamonds and circles (which in most cases need to be annotated with textual elements). Another separation can be made between *external* and *internal* DSLs: external DSLs are self-contained languages with their own abstract and concrete syntax and static semantics (Ghosh, 2011b; Cuadrado *et al.*, 2013). By contrast, internal DSLs rely on the language infrastructure of a *host language* (Ghosh, 2011b). Especially functional languages like Haskell are often used as host languages for internal DSLs (Kosar *et al.*, 2016). Some authors also refer to internal languages as *embedded languages* (Fowler and Parsons, 2011). Generally, internal DSLs only use a particular subset of the host language's capabilities (Fowler and Parsons, 2011).

As is pointed out by Mernik *et al.* (2005), it is often a matter of opinion whether a given language is close enough to a problem domain to be considered a proper DSL. The authors explain that it is possible to establish an ordinal scale that ranks various computer languages according to their specificity for a given problem domain. This way, it is up to the beholder where on this scale a computer language must rank so they will consider it a DSL. At this point, it is worth noting that in recent years authors have become more careful in this regard and now carefully

distinguish between DSLs and domain-specific modelling languages (DSMLs) (Kosar *et al.*, 2016). Challenger *et al.* (2016), for example, consider DSMLs as languages that mostly feature a graphical concrete syntax with an even higher level of abstraction (or specificity) than DSLs. However, there are others, who do not make this discrimination and use the terms interchangeably (Brambilla *et al.*, 2012, c.f., e.g.). Within the context of this thesis no distinction between the two terms will be made and they will be used interchangeably.

The creation of DSLs is one of the key activities performed by software language engineers (SLEs) (Barisic *et al.*, 2012b). In order to create a self-contained, external DSLs, language engineers must define an abstract and a concrete syntax, specify the constraints for the static semantics, select the components of the solution space, i.e. built the target platform, and devise and implement suitable model transformations.

The general process for this is summarised by Barisic *et al.* (2012b): in the first step, the target domain is analysed in order to determine which concepts must find their way into the language and how best to represent them. This step is referred to as the *domain analysis* (Ghosh, 2011b) or as *domain engineering* (Mernik *et al.*, 2005). In the next step, the conceptual design of the language is devised. This step involves the creation of a first version of the languages' abstract syntax and static semantics. In the next step, an actual implementation of the language is created. This step has been considerably facilitated by ever more sophisticated *language workbenches* (Fowler and Parsons, 2011; Völter and Benz, 2013) that automate and/or support the creation of some of the important components of the language application process. Modern language workbenches like Xtext[4] also provide support for the testing of the implemented languages (Bettini, 2016). Finally, it is crucial to find out how the targeted users experience working with the new language. It must be ensured that they are able to learn the language in an appropriate amount of time and that the language enhances the productivity of its users.

While the first steps are demanding in their own right, it appears that it is especially the evaluation step that causes major problems among language developers.

[4] https://www.eclipse.org/Xtext/

For a long time, support for the various claims of potential benefits associated with the application of DSL based approaches has mostly been of anecdotal nature (Barisic *et al.*, 2012b; Kosar *et al.*, 2012). Language engineers often neglected or even completely omitted a rigorous quantitative language evaluation (Gabriel *et al.*, 2010; Barisic *et al.*, 2012a). And even if they conducted formal evaluations, often controvertible approaches were taken (Challenger *et al.*, 2016). Kosar *et al.* (2016) conducted a systematic mapping study in which they found that reports on language evaluation were rather scarce. While recent years have seen an increased scientific effort in the field of DSL evaluation (Kosar *et al.*, 2018), there is still considerable work to be done in order to create a reliable body of knowledge. More evaluations from different fields are necessary to provide support to researchers and software developers in the application and assessment of DSLs centred software development approaches (Challenger *et al.*, 2016). As is emphasized by Barišic *et al.* (2014), a single evaluation study provides only limited informative value; in order to gain real confidence, several evaluation studies with consistent findings are required. A recent systematic literature review (SLR) study by Poltronieri *et al.*, 2021 found that DSL evaluation in terms of *usability* has generated an increased academic interest that seeks to address some as of yet unresolved problems; these problems especially concern the application of established evaluation frameworks and the reporting of detected usability shortcomings.

1.7 Aims and objectives of the research project

The purpose of the research project presented in this thesis was the design, implementation and empirical evaluation of Athos. Athos is a DSL for the domain of agent-based traffic and transport simulation and optimisation. Athos was designed to support domain experts with an easily comprehensible language that allows the specification of textual models from which executable agent-based traffic and transport optimisation simulations can be generated. In general, these domain experts only have limited knowledge in software development, so that Athos has

to meet higher standards in terms of learnability, perceivability and evolvability compared to a general-purpose language (GPL) which is normally designed for use by software development experts. In other words, the DSL must provide a moderate learning curve and allow for domain experts to quickly read, understand and write (or evolve) models. Ideally, the language *measurably* increases both the effectiveness and efficiency with which its users create these models in comparison to alternative approaches.

The contribution of this design-science study is thus twofold: in the sense of design-science (Hevner *et al.*, 2004) it provides innovative artefacts (constructs, models, methods, and instantiations) that address existing problems in the development of complex agent-based simulation software. More precisely, it provides a novel DSL (construct) that allows the creation of concise textual models based on appropriate abstractions (models) for the domain of traffic and transport optimisation and simulation. The language is implemented in a way that models are automatically validated and transformed into executable simulations (instantiation) in which powerful optimisation algorithms are leveraged (methods). The second important contribution is a thorough DSL evaluation that demonstrates how scientific evaluation protocols and techniques can be rigorously applied and combined for an assessment of the extent to which the newly created artefacts improve the current status quo. Thus, the intended gains in effectiveness and efficiency were empirically evaluated and the results are presented in this study. This way, the project did not only generate evidence in support of claims on an enhanced productivity through application of the DSL, it also provided a valuable contribution to the body of knowledge on formal DSL evaluation.

1.8 Research questions addressed in this thesis

The development of a *good* DSL is considered an intricate task because it requires knowledge from the field of software language engineering and at the same time a thorough understanding of the problem domain (Mernik *et al.*, 2005). This already

difficult task becomes even more involved, when the intended solution domain requires additional knowledge that is rarely found among SLEs. Though there are various tools known as *language workbenches* which facilitate some of the technical aspects of DSL development (Völter and Benz, 2013), conception, implementation and evaluation of a DSL that meets all of its intended purposes poses a set of important questions that need to be researched and answered. In the course of the underlying research project, the following research questions were raised and answered:

RQ 1 What are the properties that a DSL for modelling agent-based vehicle routing problems (VRPs) and traffic scenarios must possess?

- RQ 1.1 Which elements and relations among them characterise the problem domain?
- RQ 1.2 Which elements and relations among them does the abstract syntax of the language comprise?
- RQ 1.3 Which elements are required to model agent behaviour in agent-based routing simulations?
- RQ 1.4 Which semantic constraints exist in the context of the meta-model of the language?
- RQ 1.5 What properties should the concrete syntax of the language possess?
- RQ 1.6 Which elements and respective relations characterise the solution domain of the language?

RQ 2 How can models written in the DSL be transformed into executable agent-based vehicle-routing simulations?

- RQ 2.1 How can transformations be elaborated that produce semantically correct, executable simulations from DSL models?
- RQ 2.2 How can methods for solving VRPs be implemented and how can they sensibly be integrated into the simulation platform?

RQ 3 How does the usability of the DSL compare to traditional GPL based approaches?

RQ 3.1 Which general approach can be used to provide evidence for differences in the usability of the compared approaches?

RQ 3.2 How can possible differences in the usability of the compared approaches be quantified?

RQ 3.3 Which claims about the respective usability are supported by the obtained data?

1.9 Structure of the thesis

The rest of this thesis is structured as follows: Chapter 2 places this thesis into scientific context by discussing related work in terms of simulation based research into last-mile logistics and vehicle-routing problems. Moreover, it discusses agent-based modelling approaches related to this thesis and concludes with an overview on empirical studies in which domain-specific languages were evaluated.

Chapter 3 presents Athos – the DSL developed in the context of the research project from which this thesis results. In Section 3.1 and Section 3.2 the problem domain of Athos is investigated and an answer to RQ 1.1 provided. The following sections Section 3.3 and Section 3.4 further introduce Athos and its structure and thus lay the foundations that facilitate access to the subsequent language-technology oriented chapter.

Chapter 4 presents the defining language components of Athos. Section 4.1 introduces the abstract syntax of the language and hence produces answers to RQ 1.2, and RQ 1.3. Subsequently, Section 4.2 shows some important constraints discovered in the course of the aforementioned domain analysis. This way, RQ 1.4 is appropriately covered. Section 4.3 provides details on the characteristics of the concrete syntax that was devised for Athos in order to answer RQ 1.5.

RQ 1.6 and RQ 2.1 are addressed in Chapter 5 which elaborates on the intricate technical details that ensue in any attempt to answer the approached sub-questions. In Section 5.10 the implementation and integration of an evolutionary algorithm as part of Athos' optimisation library is illustrated in order to address RQ 2.2.

The third research question RQ 3 is addressed in Chapter 6 that elaborates on the preparations that were made in order to conduct a rigorous empirical evaluation of the Athos DSL. With Section 6.2 to Section 6.7, it shows possible answers addressing RQ 3.1 and item RQ 3.2. Finally, in Section 7.2.5 the results of the empirical study are presented in great detail so that RQ 3.3 is thoroughly covered.

Second Chapter

Related work

The literature review presented in this chapter is divided according to the two main contributions of this thesis: in the first part of the literature review, Section 2.1 provides an overview of literature on ABMS platforms and other tools for routing optimisation (Section 2.1.1). It also presents studies that use these platforms and tools to answer questions from the of traffic and transport optimisation domain(Section 2.1.2). In the second part of this literature review, Section 2.2 discusses research contributions from the field of formal DSL evaluation. It starts out with the findings of some recent systematic mapping studies (SMSs) (Section 2.2.1) before it focusses on existing DSL evaluation frameworks (Section 2.2.2). After that, several reports on conducted DSL evaluations are presented (Section 2.2.3). The evaluation frameworks and studies are then discussed in Section 2.2.4. Finally, Section 2.3 displays how the presented literature is linked to this thesis.

2.1 Traffic and transport simulation

2.1.1 Platforms, languages and tools related to traffic and transport simulation

NetLogo (Seth Tisue and Uri Wilensky, 2004) is the name of both an agent-based modelling and simulation environment and the modelling language used within this environment. As such, the NetLogo language is tailored for the application in

the domain of agent-based modelling. The NetLogo language combines elements burrowed from the programming languages Logo and Lisp. From the Logo language, it takes the *turtle* concept that represents an acting entity capable of roaming the environment. In contrast to Logo, NetLogo allows the definition of several thousands of turtles. Turtles move on a grid formed of *patches*. Both turtles and patches are considered *agents* that possess certain attributes and are capable of interaction with other agents. The language supports the concept of *agentsets* that allows grouping and addressing of agents based on arbitrary properties. In addition to that, the language also features control flow constructs known from procedural programming, e.g. loops or conditionals.

Seth Tisue and Uri Wilensky (2004) emphasise that a particular strength of the NetLogo language is in the definition of complex systems that exhibit behavioural patterns emerging from the actions and interactions of numerous different entities. A special kind of agent, referred to as the *observer*, is responsible for conducting the simulation process.

The NetLogo modelling and simulation environment allows for comfortable model exploration: users can inspect the state of any agent at any given time. It is also possible to track and observe the movements of turtles through the course of a simulation. With its *BehaviorSpace* tool, the platform also enables users to test how the model behaves under several different parameter settings. The interface of the simulation environment is organised in tabs. In the *code* tab the NetLogo model code is edited. The editor offers syntax highlighting capabilities that support model comprehension. In the *interface* tab, modellers are presented a visualisation of the NetLogo model. In this tab, modellers can also add additional elements like buttons, sliders and displays that can be addressed in the NetLogo code and thus allow monitoring the simulation of the model.

NetLogo is an open platform that offers various possibilities to exchange data with other applications. It also offers an *extension application programming in-*

terface (API) that can be used to extend the NetLogo language with new features implemented in Java. Another feature of the NetLogo platform is its *controlling API* that can be used to programmatically run simulations on NetLogo models.

Seth Tisue and Uri Wilensky (2004) claim that NetLogo offers an easy entry to ABMS. The NetLogo language certainly facilitates access to several mechanisms required in agent-based simulations, e.g. asking a set of agents with certain properties to perform a given action. However, modelling with the NetLogo language – especially when defining more complex models like the routing of vehicles among dynamically assigned paths – quickly becomes indistinguishable from common procedural programming which makes the language difficult to apply for users without prior programming experience. A similar argument is made by Taillandier *et al.* (2019b) who argue that NetLogo models can either only represent rather simple real-world scenarios or are too complex to be appropriate for non-expert programmers. In (Hoffmann *et al.*, 2019a) an Athos model is compared to a corresponding NetLogo model (in fact the NetLogo code is the result of the transformations applied by the Athos generator) and it is elaborated why the Athos model is considerably less complex than its NetLogo equivalent.

MATSim (Balmer *et al.*, 2009) is an agent-based traffic and transport simulation tool that calculates traffic demand from individual agent plans. In MatSim, each agent in the population possesses an individual schedule of activities. These activities take place at different locations in a network comprised of *nodes* and *links*. From the intended activities of all agents the MATSim simulator derives plans that are enacted, evaluated and modified. This way, the traffic demand of the entire agent population is calculated based on the individual plans of its constituent parts.

The utility of a plan is defined by application of a *utility function*. A utility function awards points to a plan for each performed activity and deducts points for movements through the network or late arrivals. By application of a *co-evolutionary algorithm* plan enactment, plan evaluation and plan modification are repeated until an *equilibrium state* is achieved.

The general process of setting up a MATSim simulation starts with the preparation of real-world data for the simulator. For this, MATSim provides the concept of FUSION models that can be reused or newly implemented by modellers. FUSION models parse real-world data and transform them so that they can be processed by the MATSim simulator. In the next step, an initial individual demand module (IIDM) calculates an initial traffic demand for each agent in the population based on the data from the previous step which can be merged with additional survey data or data from logit models. In the third phase, the iterative co-evolutionary algorithm optimises the initial demand which will result in an equilibrium state in which the average agent utility is stable. In each iteration, the plans are executed, attributed a utility score and a subset of plans are re-planned.

Balmer *et al.* (2009) discuss a case study in which the daily traffic in Switzerland is modelled and simulated with MATSim. The study comprises approximately 24,000 nodes and approximately 60,000 edges as well as 1.7 million facilities on which up to five activities (home, work, education, shop, leisure) can be performed. The population of the study consists of 7 million agents that create an aggregated 7 million trips. In the presented case study the MATSim simulator is run on a machine with eight dual-core CPUs with a clock speed of 2.2 GHz and a memory capacity of 22 GByte of RAM. With this hardware the MATSim simulator needs approximately 70 minutes to simulate the traffic flow of an entire day.

VISSIM (Fellendorf and Vortisch, 2010) is a proprietary, micro-level, behaviour-based traffic simulation software that applies a time-discrete approach. It focuses on the simulation of urban traffic scenarios and offers an interface for data exchange with external software applications. Among its intended users are domain-experts such as traffic engineers and transport planners who are supported by an elaborate graphical user interface (GUI). The software allows *multi-modal* simulations including various vehicle types of private and public transport.

The VISSIM simulator is comprised of three plus one building blocks: the first block contains all aspects related to the network infrastructure like lanes, connectors, parking lots and traffic light posts; the second block features elements used to

represent traffic demand, e.g. origin-destination matrices (ODMs) or routes that vehicles take to get to their destinations; the third block encompasses elements required to control the network traffic, e.g. the traffic light programs for the traffic lights found in the first block. The additional building block is responsible for information output which can take the form of a visualisation of the network or textual representation of data / information on the state of the network and its vehicles.

For the definition of the network, VISSIM relies on the concept of *links* that are merged by *connectors*. Links represent roads and just like their real-world equivalents they comprise a specified number of *lanes*. Connectors are applied to join or split links (or their respective lanes). The network built from links and connectors is further enriched by *infrastructure objects* like traffic lights (located at a specific *spot*) or detectors (covering a given spatial area). For links, there are mandatory attributes and optional attributes. Mandatory attributes are, for example, an id or a geo-spatial location. Tolls charged for driving on the road represented by the link is an example of an optional attribute.

In VISSIM various types of private transport like trucks and cars but also bikes and pedestrians can be modelled. The system also supports the simulation of public transport on pre-defined schedules. Each type of transport comes with its own set of required and optional attributes. To govern the movement of all vehicles. VISSIM uses micro-level right-of-way determination for which the exact rules are provided by the modeller (e.g. priority-to-right in absence of yield signs or traffic lights). *Vehicle actuated signal control* can be modelled with the vehicle-actuated programming (VAP) language, a DSL for responsive signal control. The concrete syntax of the language takes the form of graphical flow charts. VISSIM also supports optimisation of signal programs.

The VAP language also allows to define dynamic routing for vehicles that adapt their path through the network to the current traffic situation. To model this kind of adaptive behaviour, VAP can be used to define conditional routing decision rules. In addition to the dynamic routing behaviour, VISSIM also allows static association of

predefined routes to vehicles. It is also possible to populate a simulation with both vehicles that use dynamic routing and vehicles that follow a predefined route.

Traffic demand can be modelled through ODMs. For this, the network is divided into several distinct zones and an ODM determines the number of vehicles that travel from a source zone to a destination zone. The exact places within a zone at which vehicles start and end their journeys are modelled with *parking lots*.

The software incorporates a micro-level car-following and lane-changing model that considers *tactical driving behaviour*, i.e. drivers do not only base their behaviour on what is about to happen in the immediate future and in their vicinity; instead they plan their actions in a way that they deem likely to be beneficial in the intermediate or long run. This, for example, enables drivers to exhibit zip-merge behaviour in appropriate situations. Via a *social force* model, VISSIM also allows the simulation of pedestrian movement that can also interact with motorised traffic.

VISSIM offers sophisticated visualisation features that allow two- and three-dimensional model exploration. The level of detail of textures applied to the visualised elements can be adapted as required. Visualisations can be persisted in different formats. In addition to subjective model exploration, it is also possible to use several metrics referred to as *measures of effectiveness (MOEs)*. Travel time or vehicle speed are two examples of commonly used MOEs. The granularity of these MOEs can be determined by the modeller, from an individual vehicle, to a given type to all vehicles in the simulation.

VISSIM uses Microsoft's *component object model (COM)* as an interface to additional applications. An example in which VISSIM interacts with other software applications is presented in (Hirschmann *et al.*, 2010). The work shows how VISSIM can be used to supply data on vehicle speed trajectories to the passenger car and heavy-duty emission model (PHEM) (Hausberger, 2003) in order to simulate vehicle-emissions based on real-world emissions. The work also mentions an interaction between an adaptive traffic control system (MOTION, cf., e.g. (Busch and Kruse, 2001)) and the VISSIM simulator.

Krajzewicz *et al.* (2012) present an open-source suite for traffic simulation called SUMO (Simulation of Urban MObility). The SUMO suite offers the possibility to run traffic simulations at a microscopic level of detail. Development of this platform started in 2001 with the intention to establish a platform capable of simulating large-scale urban models. Due to considerable development effort SUMO today can be considered a highly developed fully-fledged extensible traffic simulation suite.

The SUMO suite offers several applications (tools, scripts and plug-ins) that enable or support the design, execution and exploration of traffic simulations. A coarse overview on these tools and extensions as well as on projects that leverage these tools is provided by the authors. At the heart of the SUMO suite is the *sumo* application that enables users to actually simulate (execute) the traffic models built with the support of the other SUMO plug-ins.

Based on the purpose of the respective application, the authors divide the applications provided by the SUMO platform into three different groups. The first group consists of applications that support the creation of road networks; the second group comprises applications leveraged to model traffic demand; the third group features applications used to execute the modelled simulations.

In SUMO, traffic networks are represented as graphs built of nodes and arcs. Nodes are used to represent intersections of a road network and edges are used to represent the roads of the network. Among other attributes, edges also feature right-of-way rules or traffic-lights. Edges comprise a number of lanes each with its own specification of allowed vehicle types and vehicle speeds.

As SUMO allows traffic simulation at a microscopic level, each vehicle of a simulation can be individually modelled and addressed. In order to model a vehicle, SUMO requires the definition of a vehicle id, the specification of a route that the vehicle follows, and the indication of a point in time at which the vehicle commences the specified tour. It is also possible to provide information of a higher level of granularity: SUMO allows an exact specification of the lanes to use on the roads of the path. Moreover, SUMO supports the concept of *vehicle types* that can be assigned to individual vehicle instances.

SUMO also provides applications to create multi-modal traffic simulations. In these simulations, *persons* perform a series of dependant *trips*. Each trip is associated with a certain mode of transportation so that the list of trips of a given person can comprise several different transportation modes. The trips are dependant insofar as delays in earlier trips affect the feasibility of successive trips. In other words, if an event leads to delays that have a person missing a connection mode of transport, it becomes stranded and cannot complete the list of trips. As the authors mention, currently SUMO does not allow modelling of alternative (or fall-back) strategies for persons who missed a connecting trip.

The specification of traffic demand which comprises tens of thousands of vehicles is one problem a simulation suite like SUMO needs to address. An often used mean to this end are ODMs. However, ODMs often are defined at a level of detail that is insufficient for microscopic simulations. For this reason, SUMO offers several tools that allow users to disassemble, modify and reassemble ODMs so that they feature all the details necessary to be applied within a SUMO simulation. Another way to specify traffic demands in SUMO is the definition of sources and sinks inside the network. The authors emphasise that this approach is often necessary due to the lack of publicly available data on real-world traffic demands.

SUMO runs simulations in discrete time steps. The *sumo* simulation application uses a space continuous model that stores the lane a vehicle is currently traversing together with the distance the respective vehicle has travelled on that lane. In order to simulate vehicle movement, the authors state that a car-following model is applied. Simulations can either be run with a GUI that visualises the simulation or in a headless mode. SUMO's GUI also offers some features for model exploration, e.g. it is possible to manually change pre-defined traffic-light programs in the course of a simulation run.

The SUMO software is applied to investigate several different problems within the field of traffic research. According to the authors, current vehicle communications, or vehicle-to-everything (V2X) communication, is the most important application area of SUMO. V2X is a term for technologies that allow communication

of vehicles to other vehicles and acting entities within the network as well as the infrastructure of the network. The authors state that in this line of application SUMO requires the services of a middleware that links and synchronises SUMO with software responsible for communication simulation. The authors emphasize that SUMO offers the interfaces necessary for interoperability.

Another area in which SUMO is applied is route determination analysis in the context of emerging congestion effects, a field that has gained additional interest due to the increasing pervasion of routing devices capable of adapting to the current traffic situation in a given road network. The evaluation of *traffic light programs* and *traffic surveillance systems* are two more areas of application in which researchers resort to running simulations on the SUMO platform.

The authors mention a series of features that were under development at the time the article was written. Among these features is the integration of bicycles in a way that they affect road traffic depending on the road infrastructure. Another feature the authors aim to implement is a module for the integration of real world time tables so that modelling of public transport is facilitated. The authors stress that they intend to establish SUMO as a suite used in academia for the evaluation and investigation of models and algorithms pertaining to traffic simulation and traffic management.

Taillandier *et al.* (2019a) present GAMA, an ABMS platform. They especially focus on GAML and how proficiently the GAMA platform handles spatial modelling aspects. With GAML, GAMA possesses a DSL specifically developed for this platform. GAMA is a complete integrated development environment (IDE) based upon the *Eclipse* framework[1] and consisting of several plug-ins and features that together form a coherent simulation and modelling platform. The authors also point out that with its GAML DSL the platform allows the integration, visualisation and agentification (i.e. create agents from) numerous data persisted in several different formats.

[1] `https://www.eclipse.org/`

According to the authors, GAMA was invented to overcome the shortcomings of other ABMS platforms like NetLogo[2], Repast[3] or CORMAS[4]. The authors claim that these platforms either allow only the modelling of comparably small models without too much detail or quickly become too intricate to be used by modellers who do not have a solid background in the application of GPLs. Another problem the authors point to is that many ABMS platforms offer only weak or even no support for the integration of geographic information systems (GIS) data. In GAMA, the authors emphasize, the integration (and agentification) of GIS data is well supported by the platform.

The GAMA platform is supposed to combine the strengths of its competitors. For example, the authors mention the capability of NetLogo to allow modellers to quickly switch between a modelling and a simulation view. The authors express their view that this is one of NetLogo's key features since modellers receive immediate feedback on the effects their changes to the model have on the simulation. GAMA also enables modellers to quickly modify and execute models and it even surpasses NetLogo's capabilities by offering sophisticated features like multiple visualisations of the same model or multi-simulations that allow to run and visualise several simulations in parallel.

As the newest features introduced in the presented version (1.8) of GAMA, the authors mention a set of newly introduced data types supported by the GAML modelling language which drives the specification of models for the GAMA platform. Other new features presented by the authors are the support of new data formats as well as the support of additional agent architectures. The authors also note that the platform now is capable of running different experiments, i.e. differently parameterised simulations, in parallel.

[2] https://ccl.northwestern.edu/netlogo/
[3] https://repast.github.io/
[4] http://cormas.cirad.fr/indexeng.htm

The GAML DSL adopts an *agent-oriented* paradigm, i.e. every *active* element in a simulation is an agent[5]. The language also borrows concepts from the object-oriented paradigm. Two key concepts of GAML are *agents* and *species* which assume similar roles as *objects* and *classes* do in object-oriented languages. Similar to what classes do for their objects, species define certain properties that each agent of the respective species possesses. Unlike classes, however, a species can also define properties for the set of all its instances, i.e. a species can determine properties of the *population* of its agents. A species can have a *parent species* that passes down its properties to its *children* just like *superclasses* do to their *subclasses*. While object-oriented languages also know the concept of *nested classes* (or inner classes), the semantics of a species *nested* in another *macro species* is different, since it is closer to the concept of aggregation (or composition) found in the object-oriented paradigm. The authors also mention a resemblance of the modelling language adopted by the NetLogo platform but stress that it introduces additional sophisticated programming concepts like inheritance or the notion of *nested agency* that spans multiple levels.

Depending on the language the platform uses for the specification of agent-based models, the authors group existing platforms for the specification and execution of agent-based simulations into three different categories. The first category comprises platforms that rely on GPLs for model specification (e.g. Repast[6] or Jade[7]). The second category features platforms whose models are specified by means of a textual DSL tailored towards the agent-based simulation domain (e.g. the presented GAMA platform). Finally, the third category offers platforms underpinned by graphical DSLs (e.g. AgentSheets[8]).

Taillandier *et al.* (2019b) present a tool for the GAMA platform that allows for a graphical definition of agent-based simulation models. With the presented tool, the authors aim to integrate *participatory modelling* features into the GAMA

[5]Since GAML agentifies nearly all relevant aspects of a simulation, simulations themselves are considered agents (i.e. instances) of the model species. Experiments are agents of the species experiment plan which (quite confusingly) has an abstract parent species called experiment.

[6]`https://repast.github.io/`

[7]`https://jade.tilab.com/`

[8]https://agentsheets.com/

platform. These participatory features are supposed to facilitate a collaborative model development among several stakeholders. In the eyes of the authors, it is this lack of stakeholder participation in the model-development and exploration process that prevents ABM – despite offering a vast body of literature – from assuming a more prominent role in decision-making processes of all kind.

The authors state that their objective was to design an easy-to-use graphical modelling language for the definition of descriptive agent-based models. The authors list four features that they consider key characteristics: availability of elements common in the object-oriented paradigm like inheritance and composition, distinction between possible and intended behaviour, modelling of multiple environmental layers (levels), and language elements that allow different visualisations of a model.

At the same time, they intended their conceptual language to remain close to the implementational level. For this reason, the authors decided to leverage the GAMA meta-model as the meta-model for their language. This also allows for a bidirectional mapping between their graphical modelling language and GAML, the textual language supporting the GAMA platform, so that their models can automatically be transformed into GAML-models and vice versa. With the presented tool, modellers first create a conceptual model with a concrete syntax whose elements closely resemble those found in entity-relationship and unified modelling language (UML) diagrams. Subsequently, the tool uses display dialogues to obtain further information on the properties of the modelled elements.

The authors emphasize that the graphical representation of models provides benefits in terms of communication among the various model stakeholders. They also stress that their decision to base their graphical language on the GAMA meta-model keeps the models close to the implementational level since GAML-models can directly be executed within the GAMA platform which also entails the possibility to directly execute and tests their models to see the actual effects of changes made to the model. In addition to that, the authors claim that a transition from their graphical models to the GAMA language and vice versa can be done with little cognitive effort.

In the final part of the paper, the authors shift their focus from participatory modelling to *participatory simulation*, which they define as the final activity in the modelling process in which stakeholders are given the chance to explore the model in an interactive way. They explain how GAMA – the target platform of their modelling language – supports *visualisation, interaction,* and *connectivity* which they deem key requirements for sophisticated participatory simulation.

In terms of user interaction, GAMA possesses an *event-layer* which offers users to interact with the simulation through mouse-movements and clicks in order to modify different aspects of a running simulation. *User-commands* are pre-defined by the modeller and activated by the user in the course of a running simulation. Based on the support of various *network protocols*, simulations run on the GAMA platform can exchange data with other programs (including other GAMA simulations run on different machines).

Philip Welch (2017) discusses the challenges and possible solutions of solving dynamic VRPs. The authors draw their experience from the development of a VRP planning engine named *ODL live.* The fact that a dynamic VRP features one or more aspects that are not known from the start (e.g. additional customers that demand to be timely serviced when the driver/vehicle has already set out for the tour or changing traffic conditions) requires computing a new solution whenever such new information becomes known.

The authors stress that a naive recalculation from scratch is too time consuming to be an acceptable approach for real-world systems that have to present solutions for changed situations within seconds. Instead, the authors have taken an approach that *incrementally* builds a solution for the new problem using the solution for the old problem as a vantage point. The authors state that they modified Graphhopper[9] and JSprit[10] in a way that now allows to heuristically solve even larger problems consisting of up to 1000 services within very few seconds. The presented approach is based on effectively reusing obsolete solutions.

[9] `https://www.graphhopper.com`
[10] `https://jsprit.github.io`

The authors also explain that their approach converts a dynamic problem into a static problem before applying the incremental optimisation approach. The authors then go on to list a set of modifications they had to apply on the JSprit application library in order to be able to obtain solutions for the underlying real-world problem (e.g. softened time-window constraints, even-distribution of tasks among the fleet of vehicles). They also report on a simulator they developed in order to test the ODL Live planning engine. Based on historic data, the simulator emulates the occurrence of new jobs throughout the given simulation period and also simulates the vehicles' actions and movements. The simulator uses the planning engine to obtain solutions that include the new jobs under consideration of the current simulation state (vehicle position, performed deliveries etc.). The authors place special emphasis on the importance of the capability to perform realistic offline tests on an optimisation system.

2.1.2 Simulation studies related to traffic and transport

Balmer *et al.* (2009) discuss a case study in which the daily traffic in Switzerland is modelled and simulated with MATSim. The study comprises approximately 24,000 nodes and approximately 60,000 edges as well as 1.7 million facilities on which up to five activities (home, work, education, shop, leisure) can be performed. The population of the study consists of 7 million agents that create an aggregated 7 million trips. Executed on a machine with eight dual-core CPUs with a clock speed of 2.2 GHz and a memory capacity of 22 GByte of RAM, the the MATSim simulator needs approximately 70 minutes to simulate the traffic flow for for a whole in the case study.

Quak and Kin (2020) investigated how different organisational strategies for last mile deliveries might cope with the increased demands in 2030 in terms of number of deployed vehicles, total distance covered, generated CO_2 emissions, and incurred costs. For their investigation, the authors processed real-world data from companies dealing with the delivery of groceries or parcels (either business-to-consumer (B2C)

or business-to-business (B2B)) in the area of Rotterdam – The Hague. In their calculation on the future demand for delivery services, they assumed a compound annual growth rate (CAGR) of 10 %.

The authors investigated four different scenarios in which delivery companies incorporated current trends in LML to different extents. The *base scenario* assumed the statutory implementation of zero emission zones (ZEZs) in the centres of larger cities. In these zones, only electric vehicles (EVs) were allowed to perform deliveries. However, outside of these ZEZs, the scenario presumed the deployment of conventional delivery vehicles. Half of all parcel deliveries in city centres were brought to collection points. In order to incorporate the benefits of technical advancements (e.g. smart lockers), the authors also assumed a reduction of stopping time by 20 %. The *electrification scenario* was nearly identical to the base scenario except that all vehicles were electric. In the *consolidation scenario* the difference to the base scenario was that all deliveries were brought to collection points from where they were either fetched by customers or delivered by vehicles manned with a driver-helper. The *microhubs scenario* also operated under the assumption that deliveries were only made to collection points. However, in this scenario a greater number and different types of collection points were premised.

The authors report that in the base scenario, an increase in CO_2 emissions could be observed. The authors explain that this was because the substitution of conventional vehicles by EVs in city centres could not compensate the vastly increased number of deliveries of which the vast majority was still carried out by non-electric vehicles. In the electrification scenario, CO_2 emissions induced by delivery vehicles were reduced to zero. The deployment of only EVs brought about an additional advantage: Since there were no more conventional vehicles that had to travel extra-mileage in order to avoid city centres, the total distance covered by all vehicles was reduced in the electrification scenario. The consolidation scenario reduced the total distance covered to a minimum. The authors also report that this scenario required less vehicles than the aforementioned scenarios, though they point out that this scenario heavily relies on the usage of trucks that emit higher levels

of CO_2 than vans applied in the base scenario. The microhub scenario also showed a reduction in CO_2 emissions in comparison to the base scenario though the total distance covered by all vehicles is higher than in the consolidation scenario. In this scenario, the authors also compared the usage of trucks to the deployment of vans for the delivery of parcels to the collection points.The authors found that trucks proved to be more efficient than vans. However, the authors do not generalise this result. Instead they emphasize that the more efficient vehicle type depends on the quantity of parcels to be delivered.

Regarding the costs incurred in the different scenarios, the authors state that for grocery deliveries the electrification and consolidation scenarios turned out to be the most expensive approaches. For B2B parcel logistics, only small differences in incurred costs are reported. For B2C parcel deliveries, the base scenario and the electrification scenario are reported to result in the highest costs.

2.2 Literature on empirical evaluations of DSLs

2.2.1 Systematic mapping studies on DSLs

Do Nascimento *et al.* (2012) report on a SMS they conducted for an overview on the state-of-the art in DSL development. Their focus was on identifying the domains and the extent to which DSLs are applied. Moreover, they investigated which tools, processes, methods and techniques were most often applied in the process of DSL development. Finally, they also categorize publications according to different facets. One facet was the research class of the publication, e.g. 'validation research' or 'solution proposals'. A very interesting finding of the study is that while 1142 publications were considered 'Solution Proposals' only 215 and 201 publications were classified as 'evaluation research' and 'validation research', respectively. This is in line with claims of insufficient (and idiosyncratic) efforts in terms of evaluating proposed DSLs. A general finding of the study is that from 2005 to 2010 the number of publications reporting on a DSL related topic dramatically increased.

A very current study is presented by Poltronieri *et al.* (2021). The authors present a SLR in which they investigate whether usability evaluation approaches are still a topic of interest to SLEs. Therefore, the authors set out to update a previously conducted SLR on the topic (Poltronieri Rodrigues *et al.*, 2017). In their study, they considered papers published from June 2016 to September 2020 that dealt with the evaluation of DSLs.

The authors find that usability evaluation of DSLs is a topic that draws considerable interest among SLEs. The importance of a proper usability evaluation was stressed in numerous studies investigated in the presented SLR. The authors state that in comparison to the prior SLR interest among researchers in the area even appears to have increased. Regarding the techniques employed by evaluation conductors, the number of studies who reported on the usage of quantitative data was nearly identical to those who stated to have used qualitative data. Quite a few studies even used both quantitative and qualitative data. The authors also report that most of the considered evaluation studies used a questionnaire based approach to obtain the data of interest. Only a few studies applied instruments such as logs or audio/video recordings.

The authors state that the usability evaluation of a DSL can serve to reveal problems which end users of the language might experience upon application of the language. Thus, a thorough usability evaluation can help to improve a DSL and ensure user acceptance and in conjunction the success of the language. However, the authors found that reports on research studies only scarcely broach the topic of discovered deficiencies of the language or problems encountered in the course of the applied evaluation process.

The authors summarise that in the period covered by their SLR usability evaluation even appeared to have drawn a substantially increased amount of interest among DSL researchers. One of the main shortcomings identified by the authors is that most of the inspected publications bypassed on a thourough investigation of limitations of their DSLs in terms of usability. On a similar note, many studies elided a discussion on the shortcomings observed during language evaluation. The

authors stress that even though evaluation conductors showed an increased interest in formal and standardised evaluation techniques, this topic remains one of the issues that demand further investigation since there are still no definitive standards that provide guidance to SLEs in the evaluation of their languages. The authors also point out that most evaluation studies do not apply existing usability evaluation frameworks.

2.2.2 DSL evaluation frameworks

Barišic *et al.* (2014) present a meta-model for the domain of empirical DSL evaluation. The authors thus define the core concepts and relations of the empirical DSL evaluation domain. The meta-model can be instantiated in order to analyse or prepare a DSL evaluation in a systemised way. The main contribution of the presented work is that by instantiation of the the meta-model SLEs are enabled to build a coherent body of knowledge. Drawing general conclusions from the accumulated knowledge gathered in this body of knowledge rather than from single studies allows for stronger claims and increased confidence in the obtained results.

To demonstrate the instantiation of their meta-model, the authors present an example in which they instantiate their meta-model to classify an evaluation study conducted by Kosar et al. In addition, they present an overview of three additional empirical evaluations and discuss some possible conclusions supported by commonalities of the presented studies.

Kahraman and Bilgen (2015) postulate a qualitative evaluation framework that evaluates a DSL in terms of its achieved success among its stakeholders. Their presented framework for qualitative assessment of DSLs (FQAD) was conceived to provide guidance for DSL developers towards a rigorous qualitative language assessment that determines the success of the language in consideration of what stakeholders deem beneficial.

In a first step, the authors extract from the literature (most prominently the ISO/IEC 25010:2011 standard) a set of 11 language *quality characteristics* which

might be desirable for various stakeholders. The authors also partition every characteristic into several *sub-characteristics* or *quality measures*[11] and provide a comprehensive description for all (sub-)characteristics. It is important to note that the authors acknowledge the possible existence of trade-offs between various (sub-)characteristics and thus point out the impracticality of developing a language that completely fulfils every single (sub-)characteristic.

Based on these characteristics, the authors define a formal DSL evaluation process. In this process, various stakeholders can assume the role of language evaluators. Each evaluator assigns an *importance degree* to each of the quality characteristics. The importance degree assigned to a quality characteristic determines the minimum *support level* its associated quality measures must fulfil. The evaluator then goes on and evaluates the language by attributing an actual support level to all quality measures associated to characteristics that were rated *mandatory* or *desirable*. Finally, the evaluator compares the minimum required support level to the actual support level of these quality measures. Depending on whether the language exceeds, fulfils or fails to fulfil these minimum requirements, the success level of the language is determined as either *incomplete, satisfactory* or *effective*.

The work also features two case studies in which the FQAD framework was applied to evaluate two different DSLs from a military domain. In both case studies, participants were asked to perform an evaluation of the respective DSL. Subsequent to the application of FQAD, participants were asked whether they 1) were able to sensibly use the FQAD framework, 2) found the framework beneficial in the assessment of a DSL, and 3) considered the framework an improvement over hitherto existing approaches. The authors explain that the first case study was of exploratory nature and its main purpose was to gain the necessary information to establish a stable[12] set of quality characteristics. In the second case study the authors performed the actual validation of the first stable version of their framework.

[11] While in Kahraman and Bilgen (2015) the authors mostly use the terms characteristic and sub-characteristic, Alaca *et al.* (2021) who adopt large part of this framework use the terms quality characteristic and quality measure. This thesis follows Alaca *et al.* (2021) since the former terminology better conveys the function the respective concepts have in the framework.

[12] The authors speak of finalizing the set of quality characteristics, however, even the finalized version is subjected to further optimisation and thus change.

The authors summarise that the five participants of the first case study did not report on encountering any major problems in the application of the framework. In general, participants deemed the framework useful and an improvement over their current method of evaluation. The first study led to the introduction of spreadsheets to support DSL assessment and also some changes in the terminology and more precise descriptions of quality measures. The resulting stable version was then used in the second case study. The presented results show that the three participants provided positive feedback on the applicability of the framework and its usefulness but also provided constructive criticism that motivated the authors to restructure some quality characteristics and quality measures. Following a participant's suggestion, the authors also discuss the benefits of applying the framework at the very beginning of the DSL development process to help language developers focussing on those language aspects that are important to their language's stakeholders.

Challenger *et al.* (2016) present a language evaluation framework specifically designed for DSLs from the domain of multi-agent systems (MASs). The presented framework features three dimensions: a *language dimension* and an *execution dimension* that are both of quantitative nature and also a *quality dimension* which evaluates a language in a qualitative manner. Each dimension comprises two *sub-dimensions* in which *evaluation criteria* are defined. The language dimension, for example, is divided into the *language elements* and *model transformations* sub-dimensions. The former defines the number of *abstract and concrete syntax* elements as well as the *number of constraints (static semantics)* as evaluation criteria while the evaluation criteria of the latter is the respective number of model-to-model (M2M) and model-to-text (M2T) transformations.

The framework of Challenger *et al.* (2016) is further refined and extended by the *AgentDSM-Eval* framework presented by Alaca *et al.* (2021). At the heart of this framework is an online evaluation tool which offers a series of features that support language evaluation. To evaluate their language, MAS DSML developers have to register their language and provide a description of its meta model. This

meta-model is then semi-automatically matched against a reference meta-model. From this data the tool can provide insight on the domain coverage of the evaluated language.

DSL developers are required to develop a set of case studies and upload a description for each case study. MAS developers who assume the role of language evaluators then download and work through each case study. It is important that evaluators track the time they require for each development phase. At the end of each case study, evaluators upload the measured times together with their DSML programs, the code generated from these programs, and the final code which was completed by manual code additions that could not be generated. To allow for a qualitative evaluation, evaluators have to rate several *quality measures* (partly adopted from the FQAD framework of Kahraman and Bilgen (2015)) on a five-point Likert scale.

Based on the provided data, the tool automatically calculates the values for the measures defined in the AgentDSM-Eval framework. Via its graphical interface, the tool provides averaged information on the time participants required for each development phase, the ratio of generated to manually written code, and the ratings received for the quality measures. As a unique feature, the tool also offers insight on how often each meta-model element was actually used by MAS developers. Each of these results is presented on a per-case study and also on an overall basis.

2.2.3 Evaluation studies on DSLs

2.2.3.1 Evaluations related to the concrete syntax of DSLs

Meliá *et al.* (2016) conducted a quasi experiment among 86 participants in order to determine how graphical and textual concrete syntaxes for a domain-modelling language differ in their impact on developers' effectiveness, efficiency, and contentedness with the language in the context of maintenance tasks. Participants were third-year software developers enrolled at the University of Alicante. The authors

centred their experiment around the analysability and maintainability of models. According to the authors, these are the aspects the chosen type of concrete syntax has the most direct impact on.

For their experiment, the authors defined domain models for two different applications (a course-managing and a ticket-selling system). The domain models of the two applications were defined both with a textual and a graphical concrete syntax that were based on the exact same meta-model (abstract syntax). Participants were supposed to perform two different types of tasks: *analysability* tasks, in which they had to identify and correct errors in the domain model and *modification* tasks, in which participants had to adapt the domain-model according to given instructions. For each domain the authors defined five analysability tasks and five modifiability tasks for each notation type . Every participant was assigned to one of these two domains. In both domains, half of the participants started with the graphical notation and finished with the textual notation while the other half used the notations in reverse order.

The authors used the OOH4RIA (Meliá *et al.*, 2008) approach which offers both a textual as well as a graphical concrete syntax for the same abstract syntax. Both concrete syntaxes are supported by a similar tooling. The authors took care that in their experiment the support offered by the IDE was nearly identical. This way, the authors were able to ensure that deviations in the results for each type of notation were grounded in the nature of the respective type of syntax and not in some external factor.

In their results, the authors report on two metrics with a statistically significant deviation in participants' performance that both are in favour of the textual syntax: in terms of *analysability coverage* (i.e. the portion of errors that a participant correctly identified), participants detected 73.1 % of all errors with the textual concrete syntax compared to 65.9 % with the graphical notation. The other metric in which participants exhibited a significantly better performance was *modifiability efficiency* (i.e. the number of correctly solved modification tasks per hour): with the textual syntax, participants (in theory) were able to solve 20.39 modification tasks

in an hour, with the graphical syntax participants (in theory) could only execute 14.59 modification tasks per hour. The authors also note that the numbers for all other metrics related to participants' objective performance were also in favour of the textual notation, though not statistically significant and in some cases only marginally.

In the study, participants were also asked to assess their own performance with the two different notations. Pertaining to participants' *perceived* performance, the results do not show any metric for which participants indicated in a statistically significant way that their own performance was better with either of the two approaches. The authors also measured participants' *satisfaction* with the two notations. The authors found that even though the actual performance with the textual syntax was better, participants showed a slightly higher satisfaction with the graphical approach.

From their results, the authors conclude that even though the textual notation showed to be advantageous in some objectively measured aspects, the observed benefits might not warrant the cost and effort caused by a change of the concrete syntax of a language used in a software project. The authors also stress that this is especially true in light of the higher level of satisfaction that is likely to be concomitant with the graphical notation.

Cachero *et al.* (2019) report on a study they conducted to compare the effect of a graphical and a textual concrete syntax on programmers' effectiveness, efficiency and productivity. For this, the authors conducted a quasi-experiment among 127 participants from which four groups were formed. Each group was assigned a distinct combination of concrete syntax to use and task to perform. It is important to note that the authors chose a modelling language that offered both a textual and also a graphical concrete syntax to encode a domain model. This way, the authors were able to ensure that differences in the outcome were not rooted in differences of the meta-models represented/encoded by the compared concrete syntaxes. The authors measured the percentage of several correctly defined domain-model elements

like attributes, relations, or cardinality definitions (effectiveness). In addition, the authors also counted the time participants required to complete their respective task (efficiency) and also put both in relation (productivity).

With their results, the authors are able to statistically prove that programmers' performance significantly depends on the applied form of concrete syntax. Their presented analysis of the merged data obtained for both tasks reveals that programmers were significantly more efficient in all subcategories (correctly defined attributes, operations, relations, and cardinalities) with the graphical concrete syntax. Moreover, participants that leveraged the graphical syntax were significantly faster than their peers who used the textual syntax. However, the authors also conclude that their results cannot be generalised to notations different from the ones they compared and that the decision of whether to use a graphical or a textual concrete syntax must always be made in consideration of the task to be addressed by the language.

Finally, the authors embark on explaining their results in the context of the cognitive dimensions of notations framework (CDNF). In a first step, they determine a subset of dimensions that are strongly affected by the concrete syntax of a language. Having presented the respective dimensions, they discuss how the two types of concrete syntaxes applied in their experiments affected each of the relevant cognitive dimensions. The authors consider the graphical notation to be advantageous in four out of seven relevant dimensions. In two dimensions they do not see distinct advantages for either of the two notations and in one dimension they deem the textual notation preferable.

One of the key dimensions the authors consider responsible for their results is *hard mental operations*. The authors explain that the graphical notation causes considerably less effort when determining the relations of various entities in a model than its textual counterpart which actually hampers recognition of these relations. The authors also present *error proneness* as a dimension with substantial impact on their results. In this dimension, the graphical syntax has the advantage of being an extension to UML class diagrams that are familiar to most software developers. By

contrast, the textual syntax was new to the study participants and hence they were more likely to introduce mistakes with the textual syntax. The only dimension the authors regard as being in favour of the textual syntax is *secondary notation* since the graphical notation also relies on some textual elements whereas the textual model is completely self-sufficient.

2.2.3.2 Evaluation studies of specific DSLs

Ewais and de Troyer (2014) present an empirical evaluation on three coherent graphical DSLs for the specification of adaptive three-dimensional virtual learning environments (VLEs). The three languages define different aspects of a VLE: The pedagogical modelling language (PML) is used to define the relations among the pedagogical entities of the targeted learning topic. For this purpose, PML features pedagogical relationship types (PRTs) to define a directed relation between two *learning concepts*. PRTs comprise pedagogical update rules (PURs) that are triggered when the user accesses the source learning concept of the PRT and a set of modelled conditions is met. PRTs are used to modify *user model* data as well as attributes of the learning concepts. Learning concepts are grouped in *topics*.

The adaptive storyline language (ASL) is used to model a path of *topics* that the learner is supposed to follow throughout the VLE. Storyline adaptation rules (SARs) are the linking elements used to define a directed relation between two topics. SARs allow the definition of a condition that must be met to advance from the source topic to the target topic. They also allow the definition of *adaptations* that modify learning concepts related to the respective target topic.

The adaptive topic language (ATL) is used to model adaptations within a given topic. In order to do so, ATL provides topic adaptation rules (TARs) as a directed link between two learning concepts. TARs are activated upon the occurrence of a specific event defined by the modeller. When a TAR has been activated, a guard condition must be met in order to execute the *action part* of the TAR in which the adaptations of the target learning concepts are specified.

In their pivot evaluation study, the authors intended to evaluate the usability and learnability of the three graphical modelling languages. The authors also emphasise their interest in the reception of feedback on how to improve the language. The study was conducted among 14 participants with a background in computer science. Though most participants had experience with some kind of software modelling technique, most participants were unfamiliar with the domain of 3D or virtual reality (VR) applications. Participants first were given an introduction to the three languages. After that, participants were asked to model a 3D VLE with the three languages. It is important to note that participants had to perform this task with pen and paper only, i.e. they did not have any language editor support. Finally, participants filled in a questionnaire that featured demographical questions as well as open and closed questions on their experience with and opinion on the three modelling languages. Closed questions were to be answered with a five-point Likert scale.

Closed questions that targeted similar language aspects were grouped into categories, e.g. questions regarding the appropriateness of the languages or the languages' ease of use. Based on the obtained answers, the authors associated each question with a final result that could either be 'good', 'medium' or 'poor'. The authors report to have obtained mostly positive and some neutral feedback from study participants. According to the authors, participants expressed that they regarded the languages as appropriate for the definition of 3D VLEs and felt that their expectations were met. Most participants also regarded the languages as 'self descriptive' and 'easy-to-use'. The authors also report that questions in regard to the languages' 'suitability for learning' achieved only a 'neutral' result. The understandability of the predefined PRTs used in the PMLs is reported to be a major concern among participants. In their report on the acceptability of the languages, the authors report that four of seven questions concerned with the 'perceived ease of use' received a 'good' score, whereas three questions received a 'neutral' score.

With regard to the open questions, the authors state that participants welcomed that the modelling approach used three different languages for the definition of

VLEs. It is also reported that participants expressed their appreciation for the languages' 'ease of use' and 'consistency'. Regarding specific language features, eight participants lauded that storyline adaptation rules (SARs) of the adaptive storyline language (ASL) facilitated the modelling of the storyline and four participants emphasised their appreciation for the mechanisms of the topic adaptation rules (TARs). On the other hand, there were also four participants who disliked how TARs were to be applied. Six participants are reported to depreciate the distinction between the adaptive story line (to be modelled with the ASL) and adaptive topics (to be modelled with the ATL). Finally, the authors report that they received both valuable feedback on how to improve the languages and various suggestions on how to implement the tooling for the languages.

In (de Sousa and da Silva, 2018), de Sousa and da Silva present a usability evaluation on DSL3S. DSL3S is a DSML designed for the development of spatial simulations. The language is part of an MDSD approach that generates executable agent-based simulations from DSL3S models. The implementation of the language was done by definition of a UML profile which adapts the UML for application in specific domains. DSL3S hence is a set of stereotypes and associated tag definitions for core concepts from the domain of spatial simulations (e.g. spatial variable, animat, or operation). DSL3S is supported by a tool stack which features Papyrus[13] as an editor, Acceleo[14] as a model-to-code transformation framework, and MASON[15] as the core library of the target platform.

At the heart of the evaluation study were two distinct artefacts: the first artefact was a guide that comprised 34 steps which resulted in the creation of a rudimentary predator-prey simulation. The steps started with a description of how to install the tools and then went on to show the set up of a project and the creation of the model for the simulation. The second artifact was a questionnaire which comprised three categories each featuring four questions. The first category dealt with DSL3s, the second category focused on the tool stack, and the third category targeted the

[13] https://www.eclipse.org/papyrus/
[14] https://www.eclipse.org/acceleo/
[15] https://cs.gmu.edu/~eclab/projects/mason/

general MDSD approach. All questions asked for a rating using a five-point Likert scale. The authors also tracked the profiles of the participants (e.g. knowledge in computer science or previous experience in the field of computer simulations) and the progress made within a 50-minute time window.

The authors received mixed results regarding the number of concepts used in DSL3S. The notation of the language was viewed positively but still not top-rated. The ease of learning, however, was highly appreciated by the participants. Asked for the suitability of the language for its domain, the participants again chose to only give mediocre marks. The tool stack was highly rated: Participants gave high marks to the development environment of the language (i.e. Eclipse and the plug-ins), the usability of the graphical editor and the development process in general. Especially high ratings were given to the way the language generates executable code from a model.

In the matter of how useful participants deemed the applied MDSD approach in general, participants again gave mediocre marks when asked to what extent DSL3S can support domain experts with no programming skills in the creation of their own prototypes. More participants saw potential in the language (or a similar approach) to improve communication of models to stakeholders. Users seemed not entirely convinced that DSL3S or generally an MDSD approach might form the basis for a new standard for spatial simulation modelling. Participants also appeared reluctant to apply DSL3S or a similar approach for their own projects.

De Sousa and da Silva also searched for deviation in the ratings based on either a deeper knowledge in the field of computer science or on being more experienced in the field of simulation. For this, the authors created a sub-group of those who stated to have received training in computer science and a subgroup of those who did not. Analogously, they created sub-groups from participants with and without prior experience in the field of simulation. Between the first sub-groups the only question that showed significant divergence was the question on whether DSL3S or MDSD in general could serve as a basis for a standard language for spatial simulations. Participants who received training in computer science seemed to be more open to

this idea than those who had not been trained in this field. Between the other two subgroups this question again exhibited significant divergence. Here, participants without prior experience in simulations appeared to be somewhat more appreciative towards a DSML as a new standard language for spatial simulations. Interestingly, in the question on the impression on the general development process, it was the group of participants with prior experience in simulation who consistently viewed the process more positively than the group of inexperienced participants who gave more diverse ratings in this matter.

The authors interpreted their findings in a way that suggests that missing experience with MDSD might be the reason for a negative stance towards the general MDSD approach. They also surmise that the fact that participants with prior experience in simulations took a somewhat sceptical stance on DSL3S (or MDSD) as a basis for a standard language for spatial simulations might be due to concerns of becoming constrained or losing control when relying on a DSML. The authors also point to the fact that even though experienced participants remained sceptical in the question of DSL3S as the basis for a standard language, they also appreciated the general development process that came with the language. Da Sousa and da Silva believe this to be an indication that the scepticism is more due to subjective aversion rather than factual hindrances.

In their conclusion, the authors state that participants appreciated DSL3S in general and especially its ease of learning. At the same time, the authors also note that participants appeared to remain slightly sceptical of the MDSD approach in general.

Kosar *et al.* (2010) showed that DSLs can significantly improve program/model understanding compared to the widespread adoption of application libraries for GPLs. The authors designed two questionnaires with tasks of comparable complexity: one questionnaire with tasks to be solved using a DSL (XAML) and one with tasks to be solved using an application library of a GPL (C# forms). All tasks stemmed from the domain of graphical user interface development. The tasks were designed in a way that tested how efficiently study participants could a) learn the

notation b) perceive the programs c) evolve given programs/models of the respective approach. The authors conducted their study with 36 participants. Participants received an introduction to the problem domain and training in both approaches. After participants answered the questionnaires, the authors calculated the participants' success rate for all the questions. The authors then calculated the mean success rate, standard deviation and standard error for each of the two approaches and performed a t-test on their results. The results show a significant success-rate improvement for the DSL-related tasks.

Another early and important contribution to the field of empirical DSL evaluation was made by Kosar *et al.* (2012) who presented a set of three structurally related experiments. These experiments were placed in three different domains (feature diagrams, graph description and GUIs) and each of these experiments compared a DSL to an application library or API for a given GPL. The authors investigated whether DSLs had a positive impact on participants' program comprehension in terms of correctness and efficiency. To determine participants' comprehension correctness, the authors calculated the percentage of correctly answered questions. Comprehension efficiency, was defined as the ratio of the percentage of correct answers to the amount of time required to answer the set of questions.

For each of the three domains the authors defined four application domains and for each application domain the authors developed a set of eleven questions. The questions for two application domains were set up to test participants' program comprehension using the DSL and the other two tested the same for the respective API/GPLs combination. Thus, the application domains of the DSLs were different from those of the GPLs. To ensure a structural similarity among all these question sets, the authors created a question template that all concrete question implementations were based on. The authors also asked participants for their prior experience with the respective language and inquired on how difficult participants perceived the tasks for a given application domain.

The authors defined this study using a *within-subjects* approach: for each domain, they defined two distinct groups that answered the questions from all four applica-

tion domains. The first group started with the DSL questions and then continued with the questions for the respective GPL. The second group started with the GPL questions and finished with the DSL questions. The results of both groups were combined and tested with a parameterless test for dependent samples (Wilcoxon signed-rank test).

Their results show that in all three domains application of the DSL led to significantly higher percentages in terms of comprehension correctness, i.e. effectiveness. Moreover, in all three domains participants were also significantly more efficient when using the DSLs than they were using the GPLs. In two of three domains participants had little experience with both languages and in one domain (GUI) participants were significantly more experienced using the GPL based approach. The authors express their own surprise that (despite the evidently improved results ensuing from DSL usage) on average participants perceived the use of DSLs as only slightly less complex than the use of GPLs. However, a look at the perceived simplicity separated and grouped by domain shows that in the two domains where participants had equally little prior knowledge of both approaches, DSLs where perceived as considerably simpler.

The authors provide statistical evidence which shows that in each of the three domains participants achieved significantly better results in terms of program comprehension correctness and program comprehension efficiency. Their results leave no doubt that in the domains explored by the experiments, DSLs have the potential to considerably enhance user performance. However, it is difficult to generalise these findings to other domains. It might even be possible that the test outcome is markedly different when performed by a study group with a different experience profile. For this reason, conduction of further empirical studies that either corroborate/fortify or contradict/question these findings is highly important. Only in the GUI domain, where participants were already acquainted with the GPL did participants not appear to regard either approach to be simpler than the other.

In (Kosar *et al.*, 2018) the authors replicated a modified version of their family of experiments. The first major modification of their original study was that instead of

pen and paper participants were allowed the use of IDEs for solving the tasks of the study. The second major modification was that participants no longer solved both the DSL *and* GPL tasks but either of the two. In other words, the authors changed from a *within-subjects design*, by which they compared the GPL and DSL results of the same participants, to a *between subjects design*, by which the results of different individuals using different approaches were compared. Another modification of the study design is directly connected to the IDEs allowed in the modified version: in the replication, the authors also asked for the exact IDE tools participants used in order to solve each task.

The replication study kept the approach of comparing a DSL to an appropriate GPL (supported by a library) for three different domains (feature diagrams, graph description, and GUIs). For each domain an experiment was set up in which participants had to solve a set of tasks using either the respective DSL or the GPL. As was already noted, the design of the replication study was modified and every participant was to solve either the DSL or GPL tasks for a given domain. The authors did not modify the two dependent variables: the study again investigated participants' *correctness* and *efficiency* with the respective approach.

The results show a positive effect of IDEs on the correctness and efficiency of all participants – whether they used a DSL or a GPL. The results of the replication study are consistent with the results of the original study: Throughout all three domains, the overall results indicate that participants who used the respective DSL were both significantly more effective and significantly more efficient than their peers who used the respective GPL. However, a more detailed analysis reveals that it was mainly one specific sub-category of questions that lead to these overall results: in the *evolve* question category (i.e. questions that test how well a language allows the *evolution* of existing programs) the DSLs significantly outperformed the GPLs in all three domains. The results for the other two question categories were less distinct (e.g. in the GUI domain the DSL did not significantly outperform the GPL in the *learn* and *understand* category). This is an important difference to the original study in which DSL users achieved significantly better results throughout

all question categories in all three domains. Kosar et al. attribute this to the IDE support which seems to benefit the GPLs more than the DSLs. Finally, the authors found that DSL users applied different IDE tools than GPL users. However, since there was no clear pattern in the way tool usage of the two groups differed, this might still merit further investigation.

Johanson and Hasselbring (2017) conducted an online survey that featured an integrated controlled experiment to evaluate their *Sprat Ecosystem DSL*. Sprat is a language for the specification of simulations of maritime ecosystems. The primary aim of the presented study was to analyse and compare domain experts' performance in the creation of a maritime simulation with a conventional GPL and the Sprat DSL. In their online survey, 36[16] participants performed parameterisation and data recording tasks with both C++ and the Sprat DSL so that their effectiveness (correctness) and efficiency (time required) could be measured. Participants were students and graduate experts in the marine ecology domain. Most participants had only very limited knowledge in the use of programming languages like Java and C++.

The study was conducted *using a within-subjects* design. In order to mitigate effects from learning and exhaustion, the order in which a participant was to use the two approaches was randomised so that 52.8 % of the participants started the survey with the DSL as a first approach[17]. The authors recommended completing the tasks in around four minutes but no hard deadline was set so that (barring a 30 second minimum time barrier to remove non-attempts) participants could freely choose how much time they spent solving the tasks. At any time, participants could ask the system to inform them on whether their current solution was correct. There was no explicit learning phase. Instead, participants were presented with detailed descriptions and learning examples on how to apply the language for problems similar to the respective task.

[16]Number of participants after filtering participants who worked in domains not related to ecological simulations and after removing non-attempts.

[17]The authors claim this to be a bias against their hypothesis since they consider learning effects to outweigh effects from exhaustion. As the results from the evaluation of Athos will show, this is an appropriate assumption.

The authors state that participants produced significantly more correct artefacts when using the DSL. They report on a correctness increase of 61 % and 63 % for parameter specification and data recording tasks, respectively. At the same time, participants were also substantially faster with the DSL compared to when they used the GPL. The authors also present a qualitative analysis of the language in which participants stated preferring the use of the DSL over the traditional GPL approach. Participants rated the Sprat Ecosystem DSL higher in terms of the offered abstraction level, ease of use, comprehensibility, transparency of technical details, and maintainability.

As an exploratory sub-experiment, participants were asked to modify the language environment via a Java configuration file. Though the file was written in Java, the authors note that the style was not following the usual Java conventions but rather resembled a DSL embedded into Java. Despite the fact that participants claimed to have only limited knowledge in Java development, most of them were able to complete this task successfully.

The authors conclude from their study that their Sprat Ecosystem DSL offers an appropriate level of abstraction in combination with a comprehensible and concise syntax that aptly mirrors the language of the ecosystem simulation domain. The authors also claim that the use of their DSL increases the maintainability of the simulation software.

Challenger *et al.* (2016) present an example application of their MAS DSL evaluation framework. The authors demonstrate the application of the framework by evaluation of their own *SEA_ML* language. For the abstract syntax of SEA_ML, they state (amongst other facts) that the meta-model features eight different viewpoints that assemble 67 different classes with a total of 83 attributes and 38 associations. The authors note that these numbers can also be determined for any other MAS related DSL in order to be compared to those of SEA_ML. They argue that the language with more elements is likely to be more expressive and allows for more detailed modelling.

In order to evaluate the other two dimensions, a multi-case study approach was applied. In this study, MASs for four different domains had to be developed by a study group and a control group. Each group member had to perform a complete development cycle (analysis, modelling, implementation, testing, maintenance) on four different case studies. Members of the study group used SEA_ML. Members of the control group were allowed an arbitrary combination of general-purpose modelling notations (e.g. UML), agent-development methodologies (e.g. Tropos (Bresciani *et al.*, 2004)) and manual code development for an appropriate multi-agent platform (JADEX (Pokahr *et al.*, 2013) or JACK (Winikoff, 2005)).

By analysing the results of the case studies, the authors were able to find the data for the evaluation criteria defined for the execution dimension of the framework. For example, for the *development time* criterion, which belongs to the *development* sub-dimension, the authors averaged the time participants required in each phase of each case study. They also calculated an average of the number of diagrams created by participants as well as the average number of model elements used in these diagrams (*input MAS Model* criterion, *modelling input/output* sub-dimension), averaged the number of generated files and lines of code (LOCs) (*generated MAS artefacts* criterion, *modelling input/output* subdimension), and they also calculated the ratio of generated elements to manually developed elements (*overall (output) performance ratio* criterion, *development* sub-dimension).

Their results show, that on average, 81 % of the code could be generated from the DSL models which the authors claim to be a considerable reduction in the complexity of MAS development. Regarding the development time, the study shows that application of SEA_ML accelerates the development process: on average, the study group members required only around 3.5 hours per case study, whereas members of the control group required around 6.25 hours. The DSL excelled especially in the implementation phase: where SEA_ML participants finished this phase in an average of 37 minutes, by contrast, members of the control group required nearly four times as much time (2 hours and 19 minutes). Members of the study group also outperformed control group members in the testing phase as well as in the

maintenance phase. Both of these phases were finished in roughly half the time the control group required (testing: 17 vs. 34 minutes, maintenance: 26 vs. 50 minutes).

As part of the qualitative evaluation, members of the study group on average rated the extent to which SEA_ML facilitated development with four out of five points. Asked for the main benefits of the language, study group members lauded the ease of use due to the graphical concrete syntax, the accelerated development speed and the support for performing brainstorming that results from language concepts that are close to the problem domain. The main point of criticism was that manually added code could not be automatically transferred to the original DSL model.

The evaluation framework of Challenger *et al.* (2016) is applied by Kardas *et al.* (2018) who introduce DSML4BDI, a DSML for the model-driven development of BDI MAS. The authors note that their language not only allows to model the internal structure of one single agent, it also enables users to model the entire system in which multiple agents of possibly different types interact. With the abstractions introduced by the language, the authors seek to address the complexity of BDI-based MAS development.

The authors implemented the metamodel in Ecore[18] (Steinberg, 2009) and used the Sirius framework[19] to provide a concrete syntax for the language. Model instances are transformed into Jason code via M2T transformations implemented with the Acceleo[20] transformation specification language. The Sirius framework was also used to define a set of tools that features four different types of BDI MAS diagrams (MAS, Agent, Plan, and Logical Expression)

The authors applied the evaluation framework of Challenger *et al.* to determine the benefits obtainable by basing BDI MAS development on DSML4MAS. The authors conducted a comparative evaluation study in which their language was compared to manual code development. To this end, they formed two evaluator groups each consisting of four evaluators (participants). Both groups had to develop

[18] https://www.eclipse.org/modeling/emf/
[19] https://www.eclipse.org/sirius/
[20] https://www.eclipse.org/acceleo/

an MAS based on a case study designed by the authors. Members of one group used DSML4MAS to develop this MAS and members of the other group followed a traditional approach centred around manual implementation.

The authors claim that their results provide evidence that DSML4MAS allows for substantial gains in development efficiency. As concerns the *generation performance*, the language is shown to be capable of generating considerable portions of the final system: on average, the four participants who applied DSML4MAS were able to generate 89 per cent of the LOCs of the final system. The authors emphasize that one participant successfully applied the language to generate the entire target code from the DSML4MAS input model. Regarding the *development time performance*, members of the DSML group required slightly more time in the modelling phase but were considerably faster in the *implementation* and *test* phases so that in the end, they were nearly three times faster than members of the other group. The authors report on mostly positive feedback in the qualitative evaluation together with some constructive criticism which they turned into improvements of the language and its tools.

Finally, Kardas et. al emphasize that their conducted language evaluation based on Challenger's framework is a distinguishing factor to similar DSMLs that have not yet been evaluated in a systematic way. They also claim that the formally proved benefits of the language might further the adoption of their language and thus adoption of agent-oriented software engineering (AOSE) techniques in general.

To exemplify how to use the AgentDSM-Eval framework and its online tool, Alaca *et al.* (2021) demonstrate how they used the framework for an evaluation of the Prometheus design tool (PDT) which features an MAS DSML to support the Prometheus (Padgham and Winikoff, 2005) AOSE methodology. For their empirical evaluation they developed two different case studies that were completed by 16 evaluators. Participants developed the multi-agent systems (MASs) defined in the case study twice: once with PDT and once with a GPL. The results show that evaluators completed the case study in less time with the usage of PDT. Despite falling behind in the *modelling & design* stage (PDT users spent an average of 56.1

minutes whereas GPL developers only spent an average of 35.3 minutes in this phase), this early investment of time paid off in the ensuing phases in which PDT users were considerably faster (*implementation:* 62 minutes vs. 113 minutes, *testing:* 21. minutes vs. 55.8 minutes). Regarding the performance of the generator, 38 per cent of the final software were generated code derived from the information in the PDT models. In the qualitative evaluation, PDT achieved an average score of 3.21 from a maximum of 5 (average of all developers, all case studies all quality criteria). The authors interpret this as an indicator for a general satisfaction of evaluators with PDT.

Ingibergsson *et al.* (2018) conducted a randomised controlled experimental study based on guidelines provided in the literature. The authors' intention was to provide evidence for a better readability of a DSL from the domain of robotics. Among the guidelines the authors took from the literature are the conducting of pilot studies, which they used to improve their DSL (and also to evaluate the study material) and dry runs that they applied to ensure the maturity of the material.

The authors defined three types of tasks: firstly, so-called matching tasks, in which participants were presented a textual natural language description of an element of functionality. From a set of possible choices, participants were supposed to select the fraction of code corresponding to the description. Secondly, so called "two-way" bug detection tasks in which participants were prompted to find bugs in either a DSL or GPL program. Thirdly, the authors designed what they refer to as "one-way" debugging tasks in which participants were asked to detect bugs in a GPL program that could not occur in a program written in the DSL. This way, they intended to find out whether the DSL provided an implicit improvement of usability by preventing end-users from introducing the respective type of bugs.

The study was conducted among a group of 20 participants. The group was rather heterogeneous regarding participants' vocational background: among the members of the group were industrial programmers, professors and also students. The only prerequisite for participation was a certain amount of experience in C++.

As to the results, the authors state that with regard to the matching questions, the DSL results were slightly better than those obtained for the GPL. Conversely, in the two-way debugging questions, the DSL was outperformed by the GPL. The authors explain this with one particular question in which the results for the DSL were extremely poor. For the third question type, participants had to detect two distinct bugs in a C++ program. Ten participants found one bug and three different participants found the other, no participant was able to spot both.

The authors report on a pattern that had participants becoming faster with every question for the DSL. No such pattern could be recognised for the GPL. In the summary of their results, the authors only go so far as to claim that participants could solve the tasks with both approaches nearly equally well. At this point, they emphasize that participants had no prior experience with the DSL and learned the language via *natural programming questions* during the study. The authors state that together with several possible improvements revealed by the evaluation, their DSL has the potential to increase program readability compared to programs written in a GPL, even though they did not obtain any statistically significant results for this claim from the presented study. Though participants did not prefer any of the two approaches, the authors interpret this as an indicator that their DSL may well find acceptance among developers.

Dwarakanath *et al.* (2017) introduce their accelerating test automation platform (ATAP), a tool that aims to facilitate the creation of automation test scripts for web-based systems. ATAP is driven by a DSL that features an English-like syntax. Test scripts written in the DSL are automatically transformed into Java code that uses *Selenium WebDriver*[21] for automated testing of web-based systems. The DSL also features a scoping mechanism (see, e.g. (Bettini, 2016, pp. 237 – 284), or (Völter and Benz, 2013, pp. 234)) that supports ATAP users in the definition of valid models. As an example, consider a situation where an ATAP user entered keywords that indicate that an object that can be clicked on is to follow. The scoping-mechanism automatically excludes all 'TextboxType' elements. In the underlying

[21] `http://www.seleniumhq.org/projects/webdriver/`

framework itself, such elements would be accepted at compile time but would result in run-time exceptions. On the one hand, the DSL complements the framework with additional constraints, on the other hand, the DSL would not be executable without the underlying framework. Thus, ATAP can be considered an example of the claim made by van Deursen (1997) that there are mutual benefits between DSLs and object-oriented frameworks.

Moreover, the authors claim that this symbiosis is implemented in such a way, that the end-user never has to leave the DSL level. This means the user does not only use the DSL as a mere vehicle to create Java code. Programs in the DSL can be executed directly (though in the background Java code is transparently generated) and run-time errors are linked back to the corresponding code in the DSL. It could be argued that this is just a technical detail. However, as this DSL is intended for use by domain-experts (manual testers), it is very likely that this mechanism highly benefits the DSL's usability and quality in use (Barišic *et al.*, 2011).

The authors also describe how they validated ATAP's performance in an industrial context. For this, eight project scenarios were created. These scenarios were then dealt with by three different groups of users: i) ATAP developers (tool experts), ii) automation engineers (framework experts), and iii) manual testers (domain experts). Neither automation engineers nor manual testers had any prior experience with ATAP. It was then measured how many of the scenarios could be solved by each group using ATAP. Additionally, the required time was measured. It was found that both tool experts and automation engineers could solve all of the provided scenarios while manual testers could cover the scenarios to 71 percent on average. Finally, the required time for each project was compared to an estimated time that automation engineers would need when using the framework directly. This comparison was done for automation engineers, since manual developers could not completely cover all scenarios. It was found that the time required by automation engineers was averagely 25 percent lower than the time estimated for completion of the task using the framework directly.

The authors report on some qualitative feedback given by members of the participating groups. One interesting aspect is that manual users felt overstrained and/or intimidated by the powerful yet hard-to-grasp Eclipse IDE. Analysis of the effects of the development environment on the (perceived) quality of use of a DSL is a highly interesting topic. What is more, manual testers reported that they had problems using the DSL for verification tasks. The authors assumed that one possible reason for this might be that manual testers were confronted with too many options suggested by the IDE for such tasks. This coincides with the findings of Cuadrado *et al.* (2013) who observed that internal DSL users who were unfamiliar with the respective host language felt intimidated even though RubyTL only requires rudimentary knowledge of its host language. For this reason, Dwarakanath et al. tried to mitigate the problem by splitting verification statements into several levels, each with its own (smaller) set of options. The authors did not provide an assessment of the effects of this remedy.

2.2.4 Discussion of the presented DSL evaluation frameworks and studies

Kosar et al. Kosar *et al.* (Kosar *et al.* (2010), Kosar *et al.* (2012) and Kosar *et al.* (2018)) made several important contributions to the field of DSL evaluation. The approach of comparing a DSL to an alternative baseline application library for a GPL holds merit, especially in cases where there is no alternative DSL to which a given DSL could be sensibly compared. Focussing on how *effective* and *efficient* modellers can apply two alternative approaches in a series of different tasks that aim at a language's *learnability*, *understandability* and *changeability* appears to be a very promising approach towards a reliable assessment of any novel DSL.

In all of their studies, either a within-subjects or a between-subjects design was applied. For the within-subjects design, two groups were formed that used the compared approaches in reversed order. The intention was to mitigate effects on the results caused by the order in which participants used the approaches. Learning

effects, for example, might have a positive effect on the results achieved with the second approach. The problem is that in their results, the authors do not report on the exact number of participants in each group, only on the total number of both groups. This might pose a threat to the validity of the presented results if one group consisted of considerably more participants than the other. For example, if the group that started with the GPL questions (and finished with the DSL questions) consisted of substantially more participants than the complementary group, learning effects might bias the results towards a favourable outcome for the DSL. Another problem is that there is no information on the profile of participants in each subgroup. If one of the two subgroups consists of more experienced participants than the other, this may also have an effect on the outcome.

With their replication study, Kosar *et al.* (2018) made another valuable contribution to the field of DSL evaluation. Not only does the presented study substantiate the claim that DSLs have the potential to reduce programming/modelling errors and increase the efficiency of their users, the study also shows that IDEs can benefit both DSLs and GPLs. However, while there is no doubt that the topic of IDE support merits further investigation, the study conducted within the scope of this thesis deliberately did not allow IDE usage. The reason for this decision was that the language evaluation was to focus on Athos' abstract and concrete syntax and how both compare to those of an appropriate baseline language. Allowing the use of an IDE would (as the study of Kosar et al. clearly demonstrates) have led to other factors impacting the results. For example, since nearly all IDE support constraint checks, allowing the use of IDE would have led to the *static semantics* of the languages being part of the evaluation.

The work presented by Challenger *et al.* (2016) is an important contribution to the field of DSL development. The presented framework provides criteria for the assessment of a wide range of DSL components. The *language elements* sub-dimension, for example, aims to define criteria that allow an assessment of the expressiveness of the language by analysing the number of elements (classes, attributes, nodes, references, links, etc.) of the meta-model, concrete syntax and static semantics of

a language. The *development* sub-dimension, on the other hand, seeks to evaluate the degree to which the DSL under evaluation contributes to the reduction of development complexity. To this end, it defines metrics, that give insight on the ratio of generated to manually added code. The *development* sub-dimension also takes into account the time required for each phase in the development process and thus allows an assessment of the added value in terms of efficiency.

Even though the presented framework certainly is among the most sophisticated approaches in DSL evaluation, it is tailored towards the evaluation of graphical DSLs and thus only of limited suitability for the evaluation of textual DSLs. Both the study and the framework seem to operate under the assumption that language users do mostly create syntactically and semantically correct models due to the support of a languages' static semantics. While it seems reasonable to argue that graphical DSLs are generally less prone to syntactical mistakes than their textual counterparts, it would still be interesting to gain some insight on the number or percentage of mistakes made by new language users. Other than the time required for the respective development phase, the framework seems to pay only scarce attention to the *learnability* and *perceivability* of the evaluated language.

It is also quite surprising that the study only views the number of meta-model elements in one direction when viewing a large number of meta-model elements as an indicator for a high language expressiveness. Neither does the framework seek to investigate the appropriateness and necessity of the meta-model elements nor does it take into consideration that at some point each additional meta-model element might reduce both the learnability and perceivability of a language. Although the qualitative dimension of the framework might provide qualitative evidence from which statements on the perceivability and learnability might be deducible, it would be desirable if the framework provided some quantitative criteria to this end.

With AgentDSM-Eval, Alaca *et al.* (2021) presented a sensible advancement of the framework presented by Challenger et al. Especially the online tool with its numerous capabilities that support the analysis of obtained data is a convincing addition. However, other than this technical advancement, the framework's main

contribution seems to be the addition of several quality measures that allow for a refined qualitative language analysis. Hence, regarding the quantitative evaluation of a language, AgentDSM-Eval has similar shortcomings as Challenger's framework: the quantitative evaluation measures do not necessarily allow for a statement on the expressive or generative power of the evaluated language nor do they provide a deep insight on a languages impact on a user's comprehension correctness. For example, the percentage of generated code can be raised by using verbose transformations instead of transformations that result in succinct programming code. While more verbose transformations will increase the amount of generated code, they do not increase the expressive or generative power of the language.

Unfortunately, the article does not discuss how it is ensured that the solutions submitted by language evaluators are completely correct. Since the presented data provides information on the time spent for testing, it can be assumed that the case studies contain a set of pre-defined tests the implemented MAS must pass before the case study can be submitted. However, even if these pre-defined tests are provided, these do not necessarily ensure that the developed solution is absolutely correct. Moreover, it would be interesting, if the framework gathered some data on modelling mistakes made by participants in order to gain further insight into the effect a language has on the correctness of end users. While the calculated times for each development phase might allow for statements on user's efficiency, it might also be interesting if the framework provided some insight on the relation of failed tests (or implementation correctness) to the time spent developing the system.

The validation presented by Dwarakanath *et al.* (2017) is an example of an idiosyncratic (Challenger *et al.*, 2016) and hardly reproducible approach: first of all, the authors do not provide sufficient information on how coverage within a scenario was measured. This means, it can only be guessed what the coverage percentage for a given scenario (e.g. scenario 3 was covered to 60 percent by manual testers) means. What is more, the authors do not explicitly state whether automation engineers and

manual testers received any training on how to use ATAP. Another problem is the fact that the authors compared the actual required times for a scenario to estimated times. This poses a massive threat to validity since estimations are hardly exact nor completely unbiased.

It would have been highly interesting to more deeply analyse the actual influence of the IDE on the performance of DSL users. One possibility would have been to compare coverage results when performing the scenarios offline, i.e. with pen and paper (c.f. (Kosar *et al.*, 2012)) and compare those results to a control group that uses the IDE. In addition, it would have been interesting to learn more about the effects of splitting the verification statements into several levels, so that end-users have to choose from less options when performing verification tasks.

2.3 Relation to this thesis

Section 2.1 presented literature related to the simulation and optimisation of traffic and transport problems. Section 2.1.1 focused on platforms on which such simulations can be performed. It also presented languages and tools for the creation of the models to be executed on the respective platforms. The work of Taillandier *et al.* (2019a) emphasised the need for an approach that enables domain experts to actively participate in the creation of the simulated models. Section 2.1.2 exemplified the practical relevance of using the presented platforms in the domain of traffic and transport simulation.

Athos is a language that intends to facilitate the creation of models that can be executed on a given simulation platform in order to support decision making processes. To be applied in its target domain, Athos does not have to offer the entire set of capabilities and features offered by the presented simulation platforms. Instead, it is sufficient to focus on a certain set of problems that users must be able to model with the language. The Athos generator can then automatically transform these models into models of a given target platform. Each simulation platform presented in the literature review is a potential target platform for the

Athos generator. An important problem now is to determine the exact capabilities that Athos must offer and which target platform should be chosen to implement these capabilities. One example for such capabilities that the language must offer is given by Krajzewicz *et al.* (2012) who explain how sources and sinks are used in SUMO for the population of the network with traffic in the absence of data on real-world traffic demands. Athos also allows the definition of sources and sinks in the network. It is also possible to model every vehicle instance individually. For sources it is also possible to specify probabilistic distributions that the source applies when creating new vehicles (e.g. 7 % of newly created vehicles will be of type bus, 65 % will be of type car, etc.).

The transformations of the Athos generator currently transform Athos models into NetLogo models. In other words, at the time of writing this thesis, NetLogo is the (primary) target platform of Athos. The most important reason for the selection of NetLogo was the fact that at the time the research project set out, other projects in the research group of the author were also based on NetLogo which promised the chance of synergetic effects among the researchers. NetLogo's capability of fast prototyping and its capabilities for model visualisation and inspection were also aspects that strengthened the decision to define it as the target platform for Athos. While currently NetLogo is a fitting target platform for Athos, transformations to additional target platforms are part of future work to be performed (see Section 8.3 in Chapter 8).

The introduction explained how the reduction of costs and emissions together with an improved reliability of delivery services is one of the most pressing problems to be solved in the field of LML. At the core of these distribution optimisation problem lies a family of (theoretical) academic problems known as vehicle routing problems (VRPs) (Cordeau *et al.*, 2007). Various types from the family of VRPs are shown in Figure 2.1. The figure illustrates how Athos and NetLogo have overlapping capabilities like the definition of agents and their states or the definition of visualisation aspects. However, the problems of the VRPs family are not directly supported by NetLogo and would require some sophisticated programming skills

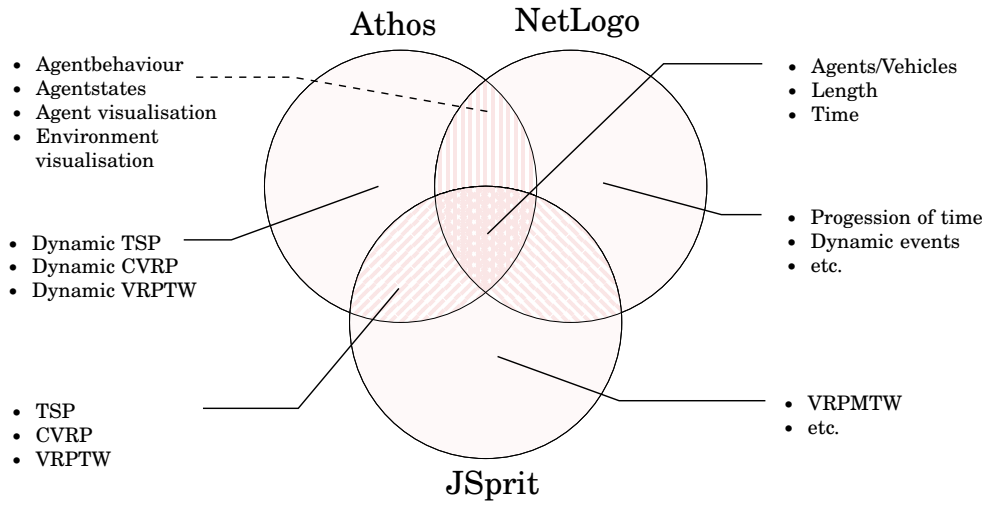


Figure 2.1: Capabilities offered by Athos, NetLogo and JSprit with NetLogo as Athos' target platform and potential support by JSprit.

to be modelled and solved in pure NetLogo. Athos itself also requires the support of a library of VRP algorithms (e.g. JSprit as shown in the figure) that are capable of solving these problems. By combining algorithms from the library with NetLogo through the transformations of the Athos generator, Athos cannot only be used to describe VRPs but it can actually provide solutions for the modelled problems. It can even introduce *dynamism* to the family of otherwise static VRPs.

Chapter 3 will start out with some additional literature and explanations on some important members of the VRP family (Section 3.1 and Section 3.2). It will then discuss how Athos is supposed to enable domain experts to participate in the creation of models via a declarative modelling approach (Section 3.3). Next, two illustrative examples will show how Athos is used to model problems from the VRP (Section 3.4) domain. The general architecture of Athos together with its target platform and supporting optimisation library will be explained in Section 3.5.

Chapter 4 will then go into full detail on implementational aspects of the most important language components. The abstract syntax of Athos is illustrated in Section 4.1 before its static semantic Section 4.2 and concrete syntax will be explained Section 4.3. Chapter 5 will focus on the Athos generator and provide very detailed explanations on how Athos models are transformed into models of the NetLogo platform. Section 5.10 focusses on how Athos can leverage implementations of optimisation algorithms.

The second part of the literature review dealt with the formal evaluation of DSLs. After the necessity for further insight on academic DSLs evaluation was carved out in Section 2.2.1, Section 2.2.2 and Section 2.2.3 presented DSL evaluation frameworks and their applications in DSL evaluation studies. The discussion in Section 2.2.4 foregrounded the approach presented by Kosar et al. (Kosar *et al.* (2010), Kosar *et al.* (2012) and Kosar *et al.* (2018)) as one of the most promising DSL evaluation approaches in the current literature on the topic. Hence, Chapter 6 will elaborate on how the approach of Kosar et al. was adopted, modified and applied in two rigorous evaluation studies centred around Athos. The results of both studies are presented in Chapter 7. Chapter 8 concludes this thesis and informs on future work on Athos.

Third Chapter

The language and its environment

This chapter discusses Athos in full detail. It starts with an analysis of the problem domain in Section 3.1 and Section 3.2. Section 3.3 then describes the general approach that Athos takes to model the problems of interest from the underlying domain. Section 3.4 then presents two concrete Athos models that provide insightful examples by which it will be discussed how Athos models are built in practice. This chapter is concluded by an architectural overview on the key components of the language and its environment.

3.1 The problem domain

The limited area of interest for which a DSL is designed is referred to as its *problem domain* (or also problem area, e.g. in (Völter and Benz, 2013; Cuadrado *et al.*, 2013)). Problem domains can be defined from a *technical* or from a *business* point of view or even both (Stahl, 2007, p. 28)[1]. The problem domain encompasses all relevant concepts like entities and processes (Ghosh, 2011b, p.4). In order to tailor a language for a specific problem domain, it is key that this domain be thoroughly analysed before language development commences. The *domain analysis* seeks an understanding of the 'ontology' (Völter and Benz, 2013, p. 162) of the analysed

[1] As this is a reference to a German book, here the original sentence along with a direct translation: 'Domänen können sowohl technisch als auch fachlich sein, wenn man diese Unterscheidung überhaupt treffen will.' (Domains can either be technical domains or business domains, if one wants to make this distinction at all).

problem domain. The outcome of such a domain analysis is a *domain analysis model*. This model is what eventually drives the implementation of the problem domain within the means provided by the solution domain (Ghosh, 2011b; Völter and Benz, 2013).

As part of the domain analysis, the *scope* of the problem domain has to be defined (Mernik *et al.*, 2005). Even though the above definition of a problem domain implies the existence of boundaries, in practice, there will often be some 'gray area' (Visser, 2008, p. 3) – especially in the fringe areas of the targeted domain. Visser (2008) points out that this 'typically leads to pressure for the DSL to grow beyond its (original) domain' (p. 3). Cleaveland (1988) emphasises the importance of foresight at this step and suggests 'to adopt an evolutionary approach and require that the features and the specification language be extensible' (p. 29). He emphasises that the definition of an optimal 'domain width' (p. 29) is of vital importance: if the domain comprises too many elements, the DSL (or its underlying generator) dilutes and will not provide any benefits over a GPL; however, if the domain is defined too narrowly, this is likely to benefit the languages' 'domain leverage' (p. 29), but almost guaranteed to render several problems unsolvable for the respective DSL.

The domain for Athos is the domain of traffic simulation and vehicle-routing and transport optimisation problems. In order to not define the intended domain too broadly or narrowly, the author of the language reserves the right to adapt the precise implementation of this rather general domain definition in a flexible dynamic way. In the current state, the language focusses on static academic vehicle-routing problems to which it allows the addition of dynamic elements like the simulation of congestion effects.

3.2 Domain analysis

3.2.1 Analysis of three basic problems

3.2.1.1 The standard travelling salesman problem

Arguably one of the most widely known routing problems is the travelling salesman problem (TSP). It is also one of the oldest routing problems: according to Larrañaga *et al.* (1999) its history dates back to the year 1759 when Swiss mathematician Leonard Euler documented the TSP. Considerable academic interest was directed towards this problem by Dantzig *et al.* (1954) and Flood (1956). The TSP belongs to the class of NP-hard problems and it has been studied more often than any other routing problem (Korte, 2008). Due to its popularity and its inherent properties, the TSP is an ideal starting point for a deeper analysis of the domain of VRPs. It features only a comparatively small set of variables and constraints, albeit being a computationally demanding problem that features the core concepts found in every vehicle routing problem (Larrañaga *et al.*, 1999).

As is shown in (Miller *et al.*, 1960), the TSP can be defined by means of a natural language formulation or as an integer programming problem. Using the natural language approach, the problem can be defined as follows:

> *A salesman seeks to find the shortest possible tour that starts at location 0 and visits each location from a set of locations indexed* 1 *to n exactly once before returning to the start location 0.*

Let 0 be the index of the starting location (also referred to as the depot). Let V be the entire set of $n+1$ locations $V = \{0, 1, \ldots, n\}$, and let V' be the set of locations different from the depot, i.e. $V' = V \setminus \{0\}$. Let the distance between any two nodes be $c_{ij}, (i, j \in V, i \neq j)$, and let subsequent stops at locations i and j be written as $x_{ij} = 1$ ($x_{ij} = 0$ is used to state that location j is not directly visited after location i). Finally, let $u_i\,(i = 1, \ldots, n)$ be arbitrary positive integers. The equivalent integer programming formulation (cf. Miller *et al.*, 1960) can then be written as follows:

$$\text{minimise} \quad \sum_{i,j \in V} c_{ij} x_{ij} \tag{1.1}$$

$$\text{s.t.} \quad x_{i,j} \in \{0,1\} \qquad \forall i,j \in V \tag{1.2}$$

$$\sum_{\substack{i \in V \\ i \neq j}} x_{ij} = 1 \qquad \forall j \in V \tag{1.3}$$

$$\sum_{\substack{j \in V \\ j \neq i}} x_{ij} = 1 \qquad \forall i \in V \tag{1.4}$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad 1 \leq i \neq j \leq n \tag{1.5}$$

Objective 1.1 defines the shortest feasible tour as the optimal solution to the problem. Constraint 1.2 states that the *decision variables* can either be zero or one. Constraint 1.3 ensures that each location has exactly one predecessor in the tour whereas Constraint 1.4 requires that every node of the tour has exactly one successor location. Constraint 1.5 was introduced by Miller *et al.* (1960) and is thus known as the Miller-Tucker-Zemlin (MTZ) constraint. It requires the tour to start and end at location 0. This way, disjoint subtours are rendered infeasible (cf. Papadimitriou and Steiglitz, 2013). Finally, it is to be noted that the presented formulation is only one of several different integer program models to represent the TSP. A total of eight of such formulations is presented in (Orman and Williams, 2007).

From the analysis of the TSP a first sketch for the meta-model (which will determine the languages' abstract syntax) and several keyword candidates (which will form the languages concrete syntax) can be obtained. Table 3.1 presents the entities and their relations found in the literature on the TSP. The first column shows the term from which keywords for the DSL should be derived. The second column presents alternative (or synonymous) terms used in the VRP domain to refer to the respective entity. The third column provides an explanation of the attributes and relations of the respective entity.

The `TSP` is the root element in the analysed part of the vehicle-routing domain. The problem is defined on a `complete Network` which is comprised of places and

Table 3.1: Elements and their relations found in the TSP.

Element	Alternative names	Description
TSP	-	The problem to be modelled and the containing root element of all other entities.
Network	Graph, Map	The structure formed by all `Nodes` and `Edges` of the network. The TSP is normally defined on a `complete` network in which all nodes are mutually connected by edges.
Node	Location, City, Place	A location with an integer `id` number that an `Agent` (the salesman) is supposed to visit.
Depot	Terminal point (Dantzig and Ramser, 1959),	A location from which the `Agent` starts and where it is supposed to finish its `Tour`.
Edge	Road, Distance, Arc	A path between two `Nodes`. Features a `length` attribute.
Agent	Salesman, Salesperson, Vehicle, Driver	The acting entity that is supposed to visit every `Node` once in the shortest possible `Tour`.
Tour	Solution, Cycle, Trip (Junjie and Dingwei, 2006)	An ordered list of `Nodes`. It is optimal for which the accumulated sum of connecting edges is minimal.

their connections. In the DSL, the places to be visited will be referred to as `Nodes` and the links between nodes will be defined using the `Edge` keyword. The acting entity of the problem will be referred to as an `Agent`. One reason why the term 'agent' was preferred to alternatives like 'vehicle' or 'driver' was the fact that the models (or programs written in the DSL) are to be transformed into various models running on different ABMS platforms for which the term agent represents the core concept. The other reason is that the word is succinct and its etymological meaning precisely matches the intended semantics of an entity that performs various actions.

Figure 3.1 shows a graphical representation of the entities and their relations. The root element is the `TSP` class. It comprises exactly one `Agent` instance (the salesman) and a `Network` built from `Nodes` and `Edges`. The TSP is considered `symmetric` if and only if the distance between any two nodes n_i and n_j $(n_i \neq n_j)$ is the same in either direction. In a complete network, there is an edge (with an associated `length`) in both directions between any two nodes in the network. The agent has access to the network. It is also associated with a tour of nodes that determines the sequence in which the nodes of the network are to be visited. The

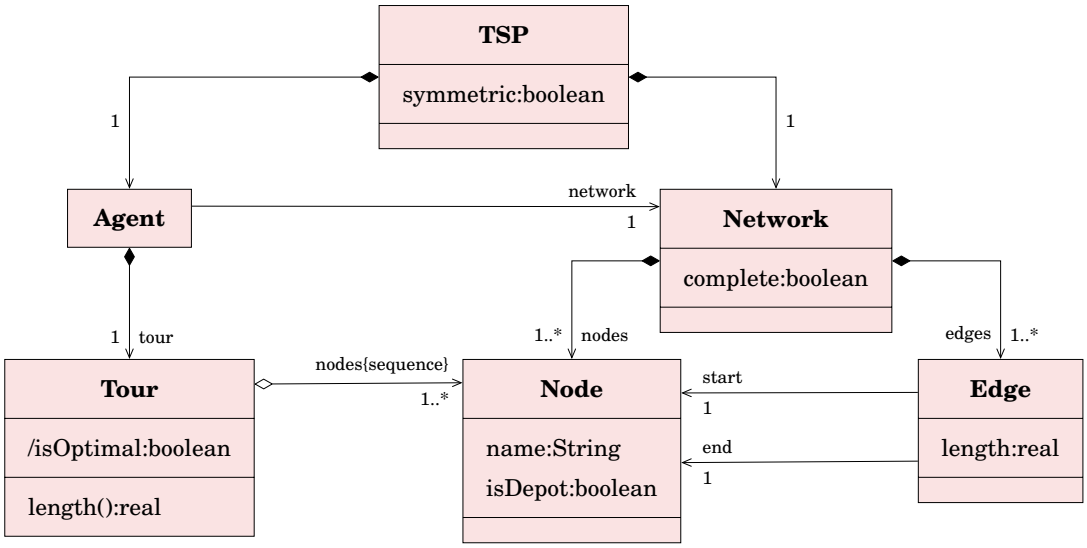


Figure 3.1: Elements and their relations found in the TSP.

tour has an associated `length` which is defined as the sum of the edge length of two subsequent nodes of the tour. The tour is `optimal` if all other feasible tours are at least as long as the optimal tour.

A formal codification of the constraints (some of which have been informally specified in the previous paragraph) that apply to the TSP is presented in Figure 3.2. The constraints are defined in the object constraints language (OCL) (cf. OMG, 2014; Cabot and Gogolla, 2012a) which allows for a precise definition of additional information on UML models such as the definitions of invariants, i.e. constraints that a model instance must adhere to in order to be considered valid.

Firstly, invariant `positiveLength` allows only edges with a length greater than zero. Invariant `noSelfTransitions` states that for each `Edge` the `start` and `end` node must be different – in other words no node is linked to itself via an edge. Invariant `completeNetwork` ensures that if the `complete` attribute of the Network is set to `true`, then for any two nodes n_i and n_j with $n_1 \neq n_2$ in the network, there must be an edge from n_i to n_j. In order to ensure that there can be no more than one edge between two different nodes, invariant `onlyOneLink` was defined. The `exactlyOneDepot` invariant states that each valid `TSP` must feature exactly one depot.

Invariant `visitAllCities` ensures that the tour comprises all `Nodes` in the `Network` so that the `Agent` visits every city of the network. Invariant `onlyOneVisit`

```
context                 Edge
inv positiveLength      length > 0

context                 Edge
inv noSelfTransitions:  start <> end

context                 Network
inv completeNetwork:    complete implies nodes->forAll(n1, n2 |
                          self.edges->exists(e|e.start = n1 and e.end = n2))

context                 Network
inv onlyOneLink         edges->forAll(e1,e2 |
                          e1 <> e2 implies e1.end1 <> e2.end1 or e1.end2 <> e2.end2)

context                 Network
inv exactlyOneDepot     nodes->select(n | n.isDepot)->size() = 1

context                 Agent
inv visitAllCities      network.nodes->forAll(n | self.tour.nodes->includes(n))

context                 Tour
inv onlyOneVisit        nodes->reject(n | isDepot).forAll(n | self.nodes.count(n) = 1)

context                 Tour
inv startAndEndAtDpt    nodes->first().isDepot and nodes->last().isDepot

context                 TSP
inv symmetric           if symmetric then
                          edges.forAll(e1 | edges.exists(e2 | e1.start = e2.end
                            and e1.end = e1.start and e1.length = e2.length
                        endif

context                 Tour::length() : real
body                    let outgoingEdges = nodes.outEdge
                        outgoingEdges.remove(outgoingsEdges->last()).length->sum()

context                 Tour::isOptimal : boolean
derive                  Tour::allInstances()->forAll(t | t <> self
                          implies t.length() >= self.length)
```

Figure 3.2: Constraints of the TSP modelled with the OCL.

then ensures that with the exception of the depot node, every node is visited only once. Invariant `startAndEndAtDpt` requires the agent to start and finish the tour at the depot node. The final invariant `symmetric` stipulates that in a symmetric TSP two nodes must be connected in both directions by edges of equal length.

The second to last element in Figure 3.2 is an operation body expression[2] that defines the length of a tour as the sum of all edges between the nodes of the tour. The final element is an expression that is to be used in order to determine whether a tour's `isOptimal` attribute is `true` or `false`. The expression states that the value is set to true only if all other tour instances have an equal or even longer length than the tour for which the attribute value is to be determined.

[2] cf., e.g., Section 7.3.6 in (OMG, 2014)

The advantage of the constraint definition via OCL over the mathematical integer program approach is that it is closer to the software development domain and thus reduces the gap between formal constraint definition and the implementation of mechanisms that ensure that models comply to all defined constraints.

3.2.1.2 The restricted multiple travelling salesman problem

Miller *et al.* (1960), present a TSP variant in which the single tour of the salesman is split into exactly t (with $t \in \mathbb{N}$) different tours that all start and end at the depot node 0. Neither of the t tours may visit more than p nodes different from the depot, yet all nodes must be visited exactly once. This variant closely resembles the multiple travelling salesman problem (MTSP) (c.f., e.g. Al-Furhud and Ahmed, 2020) except for the limitation in the number of nodes on a given tour. With the denotations defined in Section 3.2.1.1 this variant – which here is referred to as the restricted multiple travelling salesman problem (RMTSP) – can be defined as follows:

$$\text{minimise} \quad \sum_{i,j \in V} c_{ij} x_{ij} \tag{2.1}$$

$$\text{s.t.} \quad x_{i,j} \in \{0,1\} \qquad \forall i,j \in V \tag{2.2}$$

$$\sum_{\substack{i \in V \\ i \neq j}} x_{ij} = 1 \qquad \forall j \in V' \tag{2.3}$$

$$\sum_{\substack{j \in V \\ j \neq i}} x_{ij} = 1 \qquad \forall i \in V' \tag{2.4}$$

$$\sum_{i \in V} x_{i0} = t \tag{2.5}$$

$$\sum_{j \in V} x_{0j} = t \tag{2.6}$$

$$u_i - u_j + p x_{ij} \leq p - 1 \qquad \forall i,j \in V', i \neq j \tag{2.7}$$

$$1 \leq u_i \leq p \qquad \forall i \in V' \tag{2.8}$$

The Objective 2.1 of the RMTSP is directly adopted from the objective function of the TSP. Constraint 2.2, Constraint 2.3, and Constraint 2.4 are also direct adoptions of the respective TSP constraints with a slight but very important modification

concerning the latter two constraints: In Constraint 2.3 the depot is excluded from the set of of nodes that have to be the target of exactly one visiting agent, and analogously the depot is also excluded in Constraint 2.4 from the set of nodes from which exactly one agents sets out to visit a subsequent node. This exclusion is obviously necessary since there are exactly t tours that start and end at the depot, which is specified in Constraint 2.5 and Constraint 2.6, respectively[3]. The MTZ constraint was also taken from the TSP and adapted to fit the RMTSP. Constraint 2.7 on its own still ensures that there are no disjunct subtours that do not start and end at the depot. Constraint 2.7 and Constraint 2.8 together ensure that none of the tours consists of more than p nodes (different from the depot).

3.2.1.3 The capacitated vehicle routing problem

Toth and Vigo (2002) present an integer program formulation for the capacitated vehicle routing problem (CVRP) which builds upon the RMTSP. The only change is that in the CVRP every node $i \in V'$ has a demand $d_i \in \mathbb{R}^+$ (for a product or a service) and the agent has a capacity C that must not be exceeded by the accumulated demands of the nodes on any of the t tours. The integer linear program (ILP) formulation is as follows:

[3]It is to be noted that one of the constraints 2.3, 2.4, 2.5, or 2.6 could be omitted as it is an implication of the remaining three constraints (Toth and Vigo, 2002).

$$\text{minimise} \quad \sum_{i,j \in V} c_{ij} x_{ij} \tag{3.1}$$

$$\text{s.t.} \quad x_{i,j} \in \{0,1\} \qquad \forall i,j \in V \tag{3.2}$$

$$\sum_{\substack{i \in V \\ i \neq j}} x_{ij} = 1 \qquad \forall j \in V' \tag{3.3}$$

$$\sum_{\substack{j \in V \\ j \neq i}} x_{ij} = 1 \qquad \forall i \in V' \tag{3.4}$$

$$\sum_{i \in V} x_{i0} = t \tag{3.5}$$

$$\sum_{j \in V} x_{0j} = t \tag{3.6}$$

$$u_i - u_j + C x_{ij} \leq C - d_j \qquad \forall i,j \in V', i \neq j, d_i + d_j \leq C \tag{3.7}$$

$$d_i \leq u_i \leq C \qquad \forall i \in V' \tag{3.8}$$

For the CVRP the Objective 3.1 and Constraint 3.2 – Constraint 3.6 are identical to the RMTSP. The only differences can be found in Constraint 3.7 and Constraint 3.8. Constraint 3.7 can be rearranged[4] to $u_j - u_i \geq d_j - C(1 - x_{ij})$. If the agent starts at node i to visit node j, x_{ij} is set to 1 which yields $u_j - u_i \geq d_j$ or $u_j \geq u_i + d_j$. Therefore, u_i is interpreted as the accumulated demand met by the vehicle after visiting node i with $i \in V'$. Constraint 3.7 on its own eliminates subtours that do not start at the depot and it also ensures that the demand of every customer is satisfied. Constraint 3.8 asserts that the capacity C of the agent is not exceeded.

3.2.2 Specialisation and generalisation

The previous sections have shown that there is a distinct relation between the three presented problems. Throughout the literature on VRPs, this relation is referred to as *generalisation* (cf., e.g. Laporte and Nobert, 1987). Informally, a problem **B** generalises a problem **A** if it provides an additional facet that was not of explicit relevance in the generalised problem or extends the range of values allowed for an

[4] see also:
`https://how-to.aimms.com/Articles/332/332-Miller-Tucker-Zemlin-formulation.html`

existing facet. In turn, the generalised problem **A** can be considered a specialisation of the more general problem **B** in which the 'missing' facets implicitly assume a given value (or a value for which a given condition holds).

This way, the CVRP can be considered a generalisation of the RMTSP, as it explicitly adds the facets of *customer demands* and *agent capacities*. However, the RMTSP constraint which enforces that neither of the t tours contains more than p nodes can also be interpreted in terms of the added facets: if every customer i has a demand of 1, i.e. $d_i = 1 \forall i \in V'$, and the agent has a capacity of p, i.e. $C = p$, then the CVRP is equivalent to the RMTSP. In the formal ILP definition, Constraint 3.7 becomes Constraint 2.7 and Constraint 3.8 becomes Constraint 2.7.

The RMTSP is a generalisation of the of the TSP: if the number of *tours to be performed* is set to 1 and the number of *nodes allowed on a tour* is set to n, then the RMTSP becomes the standard TSP. This is because for $t = 1$ and $p = n$, Constraint 2.5 and Constraint 2.3 can be combined which yields Constraint 1.3 and the combination of Constraint 2.6 and Constraint 2.4 results in Constraint 1.4. Constraint 2.7 becomes Constraint 1.5 and Constraint 2.8 can be omitted as the number of nodes (different from the depot) equals the number of nodes (different from the depot) allowed on the single tour.

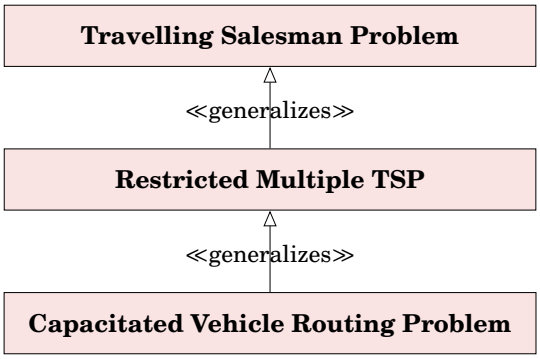


Figure 3.3: Generalisation relation between three basis routing problems.

Figure 3.3 illustrates the generalisation relation among the three problems that were discussed in the previous sections. It is important to note two aspects regarding the presented generalisation relation. Firstly, the generalisation relation is a *transitive* relation, i.e. the CVRP is also a generalisation of the TSP. And, perhaps even more importantly, the usage of the term 'generalisation' is in direct

opposition to how the term is usually used in the object oriented paradigm. Here, the RMTSP would be considered an extension or specialisation of the TSP, as its set of facets is a proper superset of the feature set of the TSP. In order to use the prevalent terminology of the analysed domain, the figure introduces the `generalises` stereotype that provides the semantics discussed in this section to the inheritance notation.

There are two important implications to the generalisation relation. The first implication is that if the DSL allows the creation of models that represent a certain target problem type, the DSL can also be used to model every problem type generalised by the targeted problem type. Moreover, as the DSL is also supposed to be transformed into a model of an appropriate solution domain, it is important to know that an algorithm from the solution domain that is capable of solving a given target problem type can also be applied to obtain solutions for problem types that are generalised by this target problem type.

3.2.3 The vehicle routing problem with time windows

In the course of the domain analysis several more publications on vehicle routing problems were searched for important concepts that needed to find their way into the abstract syntax of the Athos language. These concepts were visualised in a concept map that was created with the CmapTools software[5]. The resulting concept map is illustrated in Figure 3.4. The concept map was built in an attempt to visualise the ontology of the domain of vehicle routing problems. At some point, it was decided that a 'vantage problem' was to be chosen based on which the language was to be implemented. From the generalisation relation specified in the previous section it follows that if Athos was capable of adequate representation and simulation of this vantage problem, it would implicitly also be capable to represent and simulate all

[5] https://cmap.ihmc.us/

problems that were generalised by this problem. The vehicle routing problems with time windows (VRPTWs) then was selected as a problem suited to built the vantage point for further language extensions.

The definition of the VRPTW based on which Athos was implemented and that was also used in the training material for the evaluation study (see Chapter 6) was derived from the works of (Ombuki *et al.*, 2006; Baldacci *et al.*, 2008). Especially the work of (Ombuki *et al.*, 2006) was also of crucial importance for the implementation of an algorithm found in the optimisation library that is part of Athos' solution domain (see Section 5.10).

Let $G = (N, A)$ be a complete directed graph with $N = \{0, \ldots, n\}$ a set of $n+1$ nodes and A a set of (directed) arcs. The node with index 0 represents a depot node. The nodes $N' = \{N \setminus 0\}$ represent customers. The arcs represent connections between the nodes. Every customer $i \in N'$ has a demand of q_i units of a product, a time window $[a_i, b_i]$ within which the customer must be serviced, and a service time u_i that indicates how long the process of servicing the customer takes.

The depot hosts an infinite set of vehicles from which any number of vehicles can be deployed to service the customers. Every vehicle of the fleet has the same capacity Q. Every edge between two nodes i und j ($i, j \in N'$) is assigned a distance $c_{ij} \geq 0$ as well as a travelling duration $t_{ij} \geq 0$ that the vehicle requires to traverse the given edge.

A route or tour R is a sequence of nodes $R = (i_1, i_2, \ldots, i_{|R|})$, in which the first and the last node is the depot and which contains any customer exactly once or not at all ($i_1 = i_{|R|} = 0$ and $\{i_2, \ldots, i_{|R|-1}\} \subseteq N'$). A route is feasible, if the accumulated demand of the customers on the tour do not exceed the vehicle capacity Q, i.e. $\sum_{h=2}^{|R|-1} q_{i_h} \leq Q$, and the point in time s_{i_h}, at which the vehicle arrives at the customer i_h is before the time window of this customer closes, i.e. $s_{i_h} < b_{i_h}$, ($s_{i_h} = \max\{s_{i_{h-1}}, b_{i_{h-1}}\} + u_{i_{h-1}} + t_{i_{h-1} i_h}$ and $s_0 = 0$).

A feasible solution is comprised of a set of feasible routes $\mathscr{R} = \{R_1, \ldots, R_n\}$, that have any customer as part of exactly one feasible route.The cost or distance of a route $R = (i_0, \ldots, i_{|R|})$ is defined as $c(R) = \sum_{h=1}^{|R|} c_{i_{h-1} i_h}$.

Then the objective function can be defined as

$$\mu \cdot |\mathscr{R}| + \nu \cdot \sum_{R \in \mathscr{R}} c(R) \longrightarrow \min!,$$

with μ and ν being the weights for the number of tours (and thus the number of deployed vehicles) and the accumulated distance, respectively. Note how this corresponds to the Athos model represented in Section 3.4.1. Also see how the algorithm presented in Section 5.10.2 solves problems that comply to the given definition.

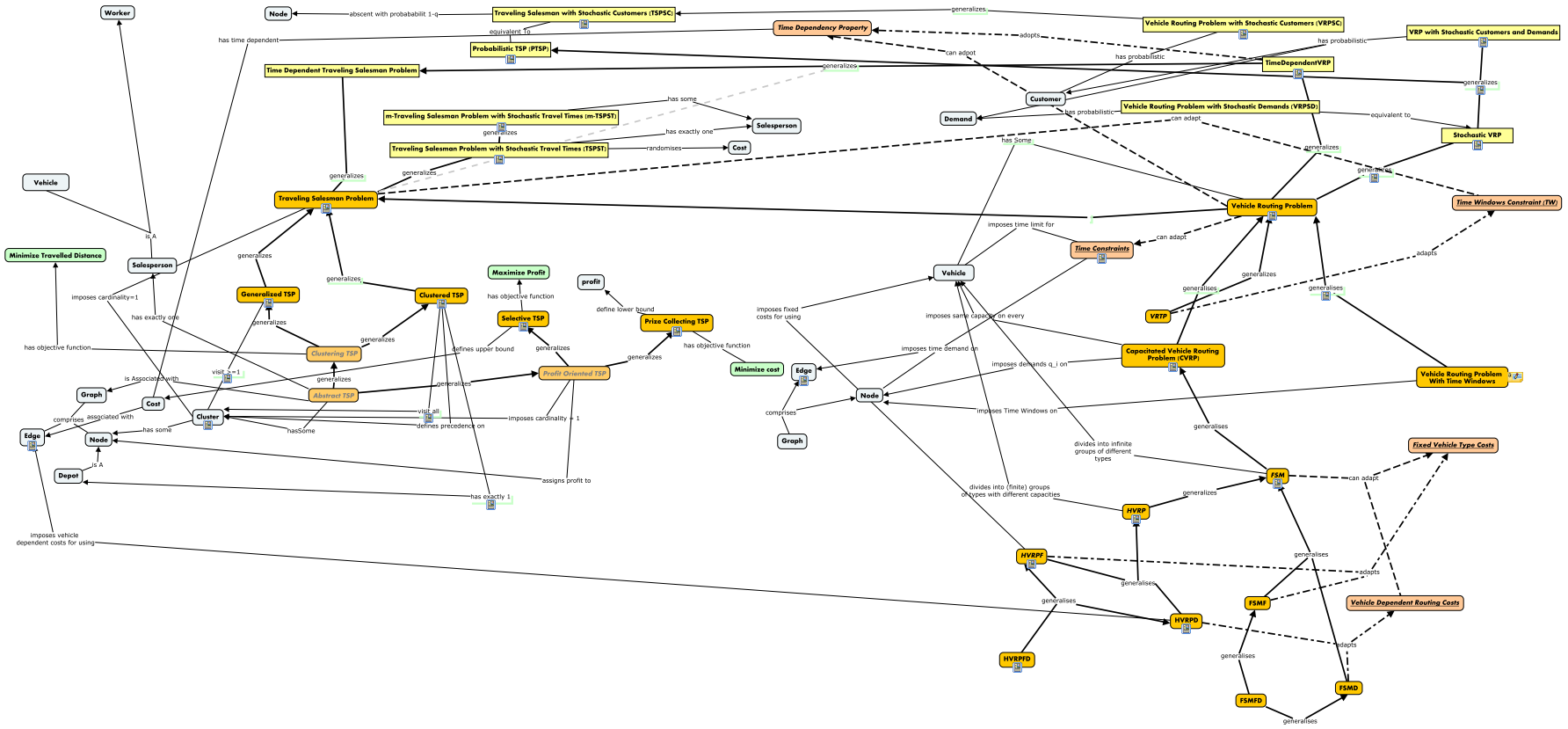


Figure 3.4: Concept map of the VRP taxonomy. **text** represents static problems, **text** represents adoptable constraints, **text** represents abstract static problems, **text** represents stochastic problems, **text** represents objective functions, **text** represents problem entities, ⟶ represents generalisations, - - ▶ represents transitive (indirect) generalisation, ⟶ represents generalisations, - - ▶ represents adaptation of constraints, ⟶ is used for any other kind of relation.

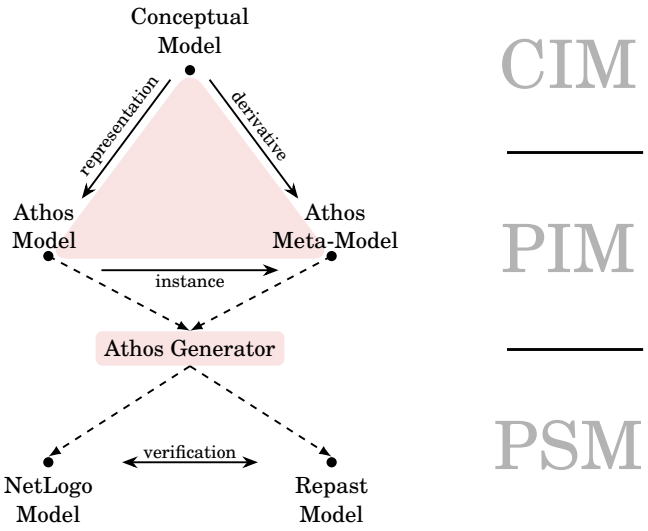


Figure 3.5: The modelling approach applied with Athos (adapted from (Hoffmann *et al.*, 2018b)): at the top are computationally independent models (CIMs), the most abstract model type. Most often, Athos model are platform-specific models (PSMs) (because they contain some computational instructions) which are transformed to platform-specific models (PSMs) for the NetLogo (or Repast) simulation platform.

3.3 Declarative textual modelling with Athos

Athos is a DSL that was designed to support domain experts in the development of agent-based traffic and transport optimisation simulations. For this the main focus of the language is on scenarios that involve vehicles (agents) that exhibit individual behaviour (e.g. finding shortest routes, replenishing stocks, etc.).

An Athos model can be comprised of a multitude of agents each of which seeking to solve one (or several) individual-level optimisation problems and acting accordingly. This way, every single agent affects its environment, i.e. the global system. Models for these simulation scenarios are defined declaratively. Thus, users do not have to think in computational (e.g. imperative) terms. Because of an increased level of abstraction, users are enable to focus on *what* to simulate rather than *how* to simulate it (c.f. Vendrov *et al.*, 2014; Borenstein, 2015).

Figure 3.5 depicts the defining language components together with how they are related. The vantage point of the approach adopted by Athos is a conceptual model derived from the domain analysis. The conceptual model reflects the ontology of the traffic and transport modelling and optimisation domain. The conceptual model is implemented as a meta-model that represents the abstract syntax of Athos.

By instantiating elements from this meta model Athos models are built. These models represent traffic or transport optimisation problems as they are found in the underlying domain.

Athos models allow the representation of aspects that are relevant in the domain (e.g., the location of a particular depot, or the capacity of roads in a given area or the extent to which a particular vehicle type contributes to road congestion). Though most Athos models are in some way related to optimisation problems (e.g., because they feature agents that need to find the fastest tour in which to visit a given set of clients), Athos only requires a declarative specification of the problem for which an optimal (or near optimal) solution is required. Athos models do not require any information on computational details on how to actually solve the given problem. Most Athos models can thus be considered computationally independent models (CIMs) (Kardoš and Drozdová, 2010, p. 90). However, Athos also features language elements that grant modellers access to the computational level (e.g. by allowing users to define parameters for a particular algorithm that is to be applied to solve a given problem; see in Section 4.3 the rules for `AgentStaticTourOptimisationBehaviour` and `AgentStaticTourEAOptimisationBehaviour`). While these model elements (and by extension the models that apply them) are more specific than their CIM level ancestors/relatives, they still do not allow any assumptions on the implementation platform. Models on this level are said to be platform independent models (PIMs) (Kardoš and Drozdová, 2010, p. 90). Every Athos model thus is a PIM (or even a CIM). Since the Athos meta-model was directly derived from the domain of traffic and routing optimisation problems (see Section 3.2 and also Section 4.1), each Athos PIM represents a concise model of the traffic and transportation optimisation domain.

The Athos generator comprises a set of transformations (see Chapter 5) that turn an Athos model into a model for a specific simulation platform like NetLogo[6] or Repast[7]. In order to be executable, these models require information that is

[6] `https://ccl.northwestern.edu/netlogo/`
[7] `https://repast.github.io/`

specific to the respective target platform which is why they are referred to as PSMs. The platform-specific details are stored in the transformations from which the generator is built. The current main target platform for Athos is NetLogo, though there is also a set of proof-of-concept transformations that can be used to transform Athos models into PSMs for the Repast Simphony platform. While Athos models remain *declarative*, NetLogo models are based on a *procedural* paradigm whereas Repast Simphony models are *object-oriented*. As is discussed by Sansores and Pavón (2005, p. 245), the capability to execute a model on different platforms (even though indirectly via transformations) brings about a crucial benefit: by ensuring that the results obtained on the different target platforms are equivalent, one important step towards model (and transformation) validation can be taken.

3.4 Athos by example

Athos is intended to be a straightforward DSL with a moderate learning curve that is manageable for software engineers and domain experts with no programming knowledge alike. As such, it appears to be a prudent approach to introduce the language with two upfront examples that highlight different aspects of the language. The first example presented in Section 3.4.1 is also the very example that was used throughout the introduction scenario and learning material of the evaluation study (see Chapter 6) and is based on the definition given in the previous section. The problem modelled in this example is a straightforward instance of a vehicle routing problems with time windows (VRPTWs). The second example presented in Section 3.4.2 demonstrates several additional features that the Athos language offers to turn static academic VRPs into their dynamic version while being capable of tracking measures of interest.

3.4.1 Example 1: The VRPTW in Athos

The example model in Listing 3.1 presents a basic VRPTW. Line 1 of the example is referred to as the *preamble*. The preamble is usually just one line in which the

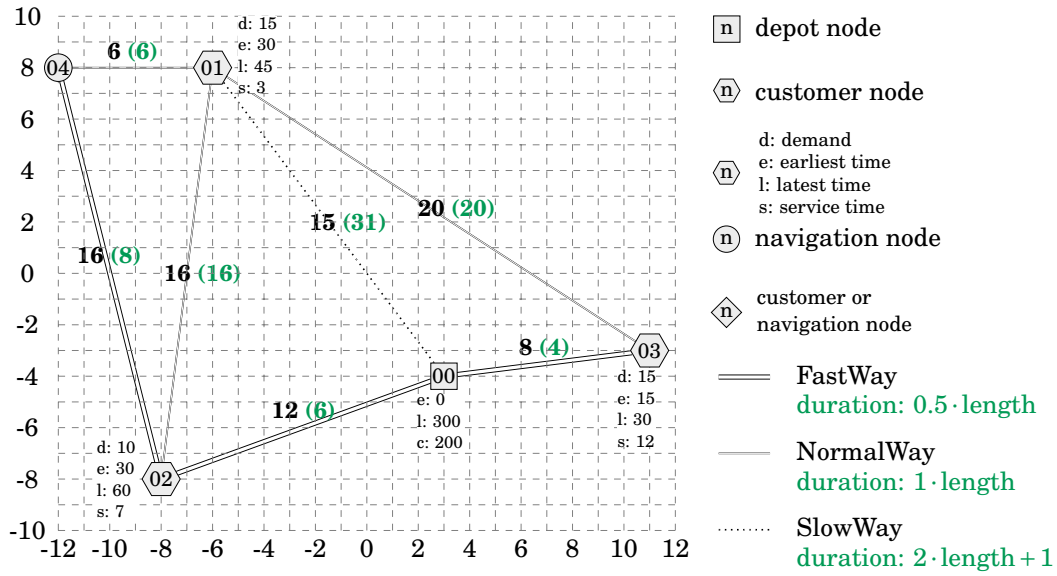


Figure 3.6: Graphical representation of the first example.

Listing 3.1: A VRPTW modelled in Athos.

```
1  model WetterauOrders1
2  products
3     foo weight 1.0
4  functions
5    durationFunction fastWayFunction  0.5 * length
6    durationFunction normalWayFunction length
7    durationFunction slowWayFunction 2 * length + 1
8  network
9     nodes
10       n0 at (3, -4) isDepot foo sprouts bar customers n1, n2, n3 at 0 latestTime 300
11       n1 at (-6, 8) hasDemand foo units 15 earliestTime 30 latestTime 45 serviceTime 3
12       n2 at (-8, -8) hasDemand foo units 10 earliestTime 30 latestTime 60 serviceTime 7
13       n3 at (11, -3) hasDemand foo units 15 earliestTime 15 latestTime 30 serviceTime 12
14       n4 at (-12, 8)
15    edges
16       s1 from n0 to n1 function slowWayFunction
17       group normalWayGroup function normalWayFunction members
18          w1 from n1 to n4
19          w2 from n1 to n2
20          w3 from n1 to n3
21       group fastWayGroup function fastWayFunction members
22          f1 from n0 to n2
23          f2 from n0 to n3
24          f3 from n2 to n4
25 agentTypes
26    agentType bar maxWeight 200
27       behaviour awt awaitTour when finished do die
28       behaviour die vanish
```

name of the model is defined. Athos does not require the name of the model to be identical to the name of the file that it is saved in. At this point it is to be mentioned that Athos is white space agnostic so that line breaks and indentations are only relevant to the modeller's convenience.

Line 2 and Line 3 form the *product section*. This section is used to define the products that agents can deliver within the network. Each product must be

associated with a *weight value* which is required so that Athos can calculate how many units of a given product an agent with a given capacity can actually transport. This way Athos can ensure that capacity constraints are not being violated by solutions provided by its underlying optimisation library (see Section 5.10).

The *functions section* ranges from Line 4 to Line 7. This section specifies expressions that can later be associated with edges (or entire groups thereof) to determine the amount of time it takes an agent to traverse the respective edge. Though currently not supported, later versions will also allow expression in terms of speed (instead of time) through the specification of `speedFunctions`[8].

The area from Line 8 to Line 24 is the *network section* (see Section 4.1.1). In Athos, the network is the environment in which the agents are embedded and within which they exhibit their behaviour. A network comprises an arbitrary number of *nodes* that are connected via *edges*. Nodes are specified in the *node section* (Line 9 to Line 14) and edges are modelled in the *edge section* (Line 15 to Line 24).

The node section in this example is composed of three different types of nodes. Line 10 defines a depot that stores an unlimited quantity of product `foo`[9] and hosts an unlimited fleet of `bar` agents (discussed below). The depot is supposed to deliver its product to customers `n1`, `n2`, `n3` that are specified as *demand nodes* in Line 11, Line 12, and Line 13, respectively. Each of these demand nodes specifies the demanded product, the demanded quantity, a *time window* (composed of an `earliestTime` and a `latestTime`), and a `serviceTime` that determines the amount of time the agent will need to remain at a particular customer node to complete the delivery. Since node $n4$ does neither serve as *source* from which agents populate the network nor as a customer / demand node, it serves only as a joint between two edges. In this thesis these kind of nodes are referred to as *navigation nodes* (also see Figure 3.6).

[8] These functions are defined in Athos' abstract and concrete syntax, but the generator currently does not support these type of functions.

[9] As it is common practice throughout many programming languages, the metasyntactic variables 'foo' and 'bar' will be used in the examples of this thesis.

Line 16 defines a single edge that connects nodes `n0` and `n1`. This edge is associated with the `slowWayFunction` that was defined in Line 7. The semantic of the expression is that the time that an agent requires to traverse the edge is equal to two times the length of the road plus an additional unit of time. As the common atomic unit of time used in agent-based simulation is a *tick*, this term will also be used in this thesis. Figure 3.6 illustrates that the length of the edge from `n0` to `n1` is 15[10] so it follows that an agent that seeks to travel from `n0` to `n1` via edge `s1` will require 31 ticks to do so.

In the *agent section* which ranges from Line 25 to Line 28, the types of agents that will populate the network are defined. In this example, there is only one single type of agent: the *vehicles* agent type is defined to be able to carry a cargo of 200 weight units[11] (Line 26). Subsequently, Line 27 and Line 28 define the behaviour of that agent type in terms of *behaviour states* (see Section 4.1.2). It is to be noted here that the first behaviour specification always represents the *starting behaviour state* of the agent. In order for an agent to switch to another behaviour state, a condition must be met that triggers a transition into the new state. In the given example, the agent type is defined to wait at a *depot* until it is assigned a *tour*. When the tour is completed, i.e. the current state is *finished*, the agent is modelled to simply leave the simulation.

In this example, the *duration* an agent travels on an associated edge depends solely on the *length* of the edge. This implies that the agents will not mutually effect each other in terms of travel times, i.e. the traversal duration for any given edge will be independent of the number or type of agents currently moving on that edge. As was mentioned in the introduction of this thesis, it is a characteristic feature of agent-based models (and of the complex adaptive systems that they represent) that agents can exert influence on each other. In a traffic network, the most important way in which agents mutually affect each other is through the occurrence of congestion

[10] All values in the illustration are rounded to the nearest integer, though in the given example the length of the edge that connects $n0$ and $n1$ located at $(3,-4)$ and $(-6,8)$ is exactly 15.

[11] The reason for the usage of 'maxWeight' instead of 'capacity' as a keyword here is that later versions are planned to also support volume limitations which would render the term 'capacity' ambiguous.

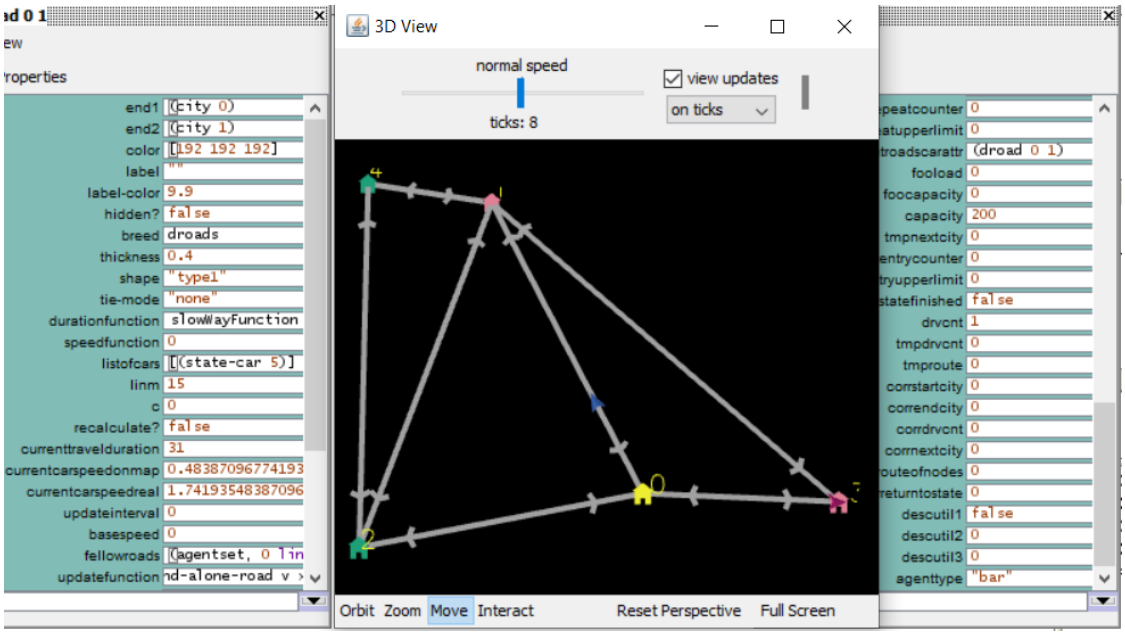


Figure 3.7: Three-dimensional view of the generated platform-specific NetLogo model.

effects. If too many agents travel on a given edge, they have to move more slowly. To model this, functions can be modified to feature a *congestion factor* that slows the movements of agents on the respective edge.

Line 17 to Line 20 define an *edge group*. Edge groups are a language element used for convenience purposes and to create more concise network models: they allow to form groups of edges that share one or several features. The common features then need only be assigned to the group (instead of every single edge). If necessary, it is possible to override any feature for a single edge even when it is a member of an edge group.

Figure 3.7 presents a screenshot taken from the NetLogo simulation that the Athos generator generates from the presented program. The middle part of the figure shows the three-dimensional view that is NetLogo's visualisation of the generated platform-specific NetLogo model. Though the edges cannot be differentiated since they are all represented as grey lines (with an attached arrow head to indicate their direction), the structure of the network clearly resembles the structure depicted in Figure 3.6.

The left hand side of Figure 3.7 shows a property view for the edge connecting `n0` and `n1`. It is to be noted here that (if not specified otherwise), Athos generates two diametrical, directed edges for every edge modelled in the Athos program. The properties view also shows that the edge's associated `durationFunction` is a `slowWayFunction` and that the edge has a `linm` (length) of 15 units from which a `currenttravelduration` of 31 ticks results.

The right hand side of Figure 3.7 shows the properties view of the agent that currently travels on the edge that was just discussed. It can also be seen that this agent has a `capacity` of 200 units, its current state is not `finished` and that it is of type `bar`. Other than `bar` there was no type defined in the example program and no actual instances were defined. The underlying optimisation algorithm applied by Athos for the modelled problem (see Section 5.10.2) provided a solution that suggested the deployment of two agents. This is why there are two agent instances to be seen in the illustration. The one whose properties were just discussed, and another agent in the bottom right-hand corner servicing node `n3`.

3.4.2 Example 2 additional elements

3.4.2.1 From static to dynamic problems

The previous example modelled a static VRPTW in which customers were to be serviced within a given time-window by a fleet of agents. The model was completely static since there were no additional agents exerting influence on the vehicles. And even if there would have been additional agents, they would not have affected each other as there were no congestion factors considered in the `durationFunctions` defined for the edges of the network. To lay the foundations for a more dynamic model, this is now done for the functions defined in Listing 3.2. The function in Line 6, Line 7, and Line 8 now feature an `accCongestionFactor` element. Together with the new agent types `baz` and `qux` defined in the agent section starting at Line 38, agents that are on the same edge will now affect one another in that

Listing 3.2: A VRPTW modelled in Athos.

```
model example1Revisited [xmin -12 xmax 12 ymin -10 ymax 10]
products
  foo weight 1.0
  fred weight 2.0
functions
  durationFunction fastWayFunction length + 9 * accCongestionFactor
  durationFunction normalWayFunction 2 * length + accCongestionFactor^2
  durationFunction slowWayFunction 4 * (length + accCongestionFactor)^3
  durationFunction noCongestion length
network
  nodes
    n0 at (3, -4)  isDepot foo sprouts bar customers n1, n2, n3 latestTime 135
    n1 at (-6, 8)  hasDemand foo units 15, fred units 4
                   earliestTime 30 latestTime 45 serviceTime 3
    n2 at (-8, -8) hasDemand foo units 10 timeWindow 30, 60 serviceTime 7
    n3 at (11, -3) hasDemand foo units 15 earliestTime 15 latestTime 30 serviceTime 12
    n4 at (-12, 8) sprouts  qux route for rte n3, n1, n2 probability 50
                         orr baz route for rte n5, n2, n1 probability 50
                             frequency 1 every 2 until 20
    n5 at (4, -3) sprouts 1 qux route for rte n3, n1, n2 probability 50
                         orr qux route for rte n5, n2, n1 probability 50 at 1
    n6 at (-10, -10) sprouts 1 baz route for rte n1, n3, n0 at 2
    n7 at (-9, -9)   sprouts 1 qux at 3
  edges
    s1 from n0 to n1  function slowWayFunction [type3, ultraThin, red]
    group normalWayGroup  function normalWayFunction members [type2, ultraThin, lightBlue]
       w1 from n1 to n4
       w2 from n1 to n2
       w3 from n1 to n3
    group fastWayGroup  function fastWayFunction members [thin, darkBlue, type3]
      f1 from n0 to n2
      f2 from n0 to n3
      f3 from n2 to n4
    group slipRoadGroup function noCongestion members [thin, lightGreen, type1]
      -> sr1 from n5 to n0
      -> sr2 from n6 to n2
      -> sr3 from n7 to n2
agentTypes
  agentType bar maxWeight 200
    behaviour awt awaitTour when finished do die
    behaviour die vanish
  agentType baz congestionFactor 0 maxWeight 150
    behaviour rte route perInstance when finished do leave
    behaviour leave idle for 300 when finished do van
    behaviour van vanish
  agentType qux  congestionFactor 10 maxWeight 150
    behaviour goToDepot route n0 when finished do loadStock
    behaviour loadStock loadCargo fred units 100 when finished do deliverGoods
    behaviour deliverGoods deliver n1 products fred when finished do rte
    behaviour rte route n2 n3 n1 repeat 4 times when finished do leave
    behaviour leave vanish
metrics updateRate 10
    for bar
      individual metric intendedRoute when notYetSet? set intendedTour
      class metric distanceCovered when  isAtCustomer?
          add distanceTo last customer
      class metric ticksEarly when isAtCustomer? and earliestTime > currentTime
          add earliestTime - currentTime
      class metric ticksLate when isAtCustomer? and latestTime < currentTime
          add currentTime - latestTime
      class metric windowsViolated when isAtCustomer? and latestTime < currentTime add 1
      class metric windowsMet when isAtCustomer? and currentTime <= latestTime add 1
    for baz
      individual metric bazRoute when notYetSet? set intendedTour
    for qux
      individual metric quxRoute when notYetSet? set intendedTour
```

they will reduce their movement speed. In this context, it is also shown that Athos allows the definition of several mathematical expressions in its function definitions, the `accCongestionFactor` in Line 7 was *squared before*[12] it was *added* to the result of *multiplying* the length with a factor of two. Because of the *brackets,* the `accCongestionFactor` in Line 8 is first added to the `length`, then the sum is *cubed* before the result is multiplied by four.

The additional agents are now sprouted at different nodes in the network. Node `n4` in Line 17 will *continuously* `sprout` agents into the simulation. More precisely, it will `sprout` 1[13] agent `every` 2 ticks `until` 20 ticks have passed. There is also a random element with regard to what agent will be created. With a 50 percent chance will the created agent be a `qux` agent visiting the nodes `n3`, `n1`, `n2`. Alternatively, the created agent will be a `baz` agent headed for nodes `n5`, `n2`, and `n1`. Note that because of the `route` behaviour, agents of type `baz` do only follow a tour without carrying out any deliveries. Also note that the agents of type `qux` can only deliver `fred` to customer `n1` if they first go to depot `no` and fill their stock of `fred`.

Node `n5` in Line 20 will sprout a `qux` agent, but the route the sprouted agent will take is randomised. This node will also only sprout exactly one agent `at` tick 1. Node `n6` in line Line 22 will sprout an additional `baz` agent `at` tick 2. Finally, `at` tick 3 node `n7` will sprout a `qux` agent that delivers `fred` to `n1`. From the definition of the `qux` agent that spans from Line 46 to Line 51 it can also be seen that all instances of this agent will start an additional `route when` their first route is `finished`.

The edges are mostly unchanged with respect to the first example. From Line 34 to Line 37, however, a new `group` of edges was defined. These are used to connect some of the new source nodes to the network and their associated `durationFunction` is not subject to any congestion effects (Line 9. With the '->' symbol used in Line 35, Line 36, and Line 37 the edges are specified as being *directed* edges or *arcs*.

[12] Athos respects the common operator precedence.

[13] The `frequency` keyword is somewhat confusing here and will be replaced in a future version of the language.

3.4.2.2 Metrics tracked throughout the simulation

From Line 52 to Line 66 a *metrics section* has been added to the program. This feature has been presented in (Hoffmann *et al.*, 2019b). The metrics section allows to define either `individual` or `class` metrics. The former is used to track data for every *single* agent of the respective agent class. The latter are used to track and accumulate data for the entire agent *class*. As their definition is not explained at great length in Chapter 4, they shall be briefly discussed in the context of this example.

The metric section must always define an `updateRate` at which the current values calculated for the defined metrics are to be output to the simulation console and written to the simulation log file. The definition of frequent metric updates might negatively affect the the speed at which the simulation runs, so if users are only interested in the results at the end of a simulation, a high update rate should be set here[14]. Having set the update rate, the actual metrics are to be defined. Metrics are always defined `for` specific types of agents.

Every metric definition is comprised of three parts that answer a specific question: *what* agent level is the metric supposed to be calculated for, *when* must a calculation be performed and *what* calculation is to be performed. For every metric the *level* (individual or class) must be specified to answer the first question. After that, a name must be provided for the metric. Next, a boolean expression must be provided to define when the calculation is to take place. Currently, these expressions should always contain either the expression `isAtCustomer?` to indicate that the calculation is to be performed when the agent arrives at a customer or `notYetSet?` to indicate that the metric is to be calculated when the agent is born. Finally, the exact calculation to be performed is specified with an expression that follows either the `add` or `set` keyword. With `add` a cumulative metric is specified, with `set` it is specified that the value of the metric is updated to the value the respective expression evaluates to.

[14]Future versions might introduce the keyword ‘summary’ here, so that the metrics are only output at the very end of a simulation.

As an example, the metrics defined for the `bar` agent from Line 53 to Line 62 provide the following information to the language user when the simulation is executed: the individual `intendedRoute` metric will display the sequence of nodes the agent intends to visit. The class metric `distanceCovered` defined in Line 55 and Line 56 provides information on the total accumulated mileage of all `bar` agents. The agents update this value whenever they arrive at a customer in their list.

Of special interest are the metrics related to time windows defined from Line 57 to Line 62. These are also updated whenever a `bar` agent arrives at a customer. Which metric is updated how depends on the circumstances under which the respective agent arrives at a customer. For example, if an agent arrives before the time window of the customer commenced, then the metrics `ticksEarly` and `windowsMet` are updated. For the `ticksEarly` metric, the difference between the tick at which the time window of the customer opens and the tick at which the agent arrived at the customer is added to the current value of the metric. For the `windowsMet` metric, the current value (which by default starts at 0) is increased by one so that this metric acts as a counter. The metrics `ticksLate` and `windowsViolated` are updated analogously should the agent arrive at the customer after the time window was closed.

3.4.2.3 Altered visualisation of elements

The last notable difference to the program of example 1 is that in this second example, there are several parts in the program at which *appearance specifications* are defined. Appearance specifications are used to alter the standard design of different elements in the simulation. They are always specified within a pair of square brackets to emphasize that they do not alter the *behaviour* but only the *visualisation* of the generated simulation. The appearance specification in Line 1, for example, extends the borders of the generated simulation. The appearance specifications in Line 25, Line 26, Line 30, and Line 34 are used so that the generated edges can easily be distinguished by the way they are visualised in the generated simulation.

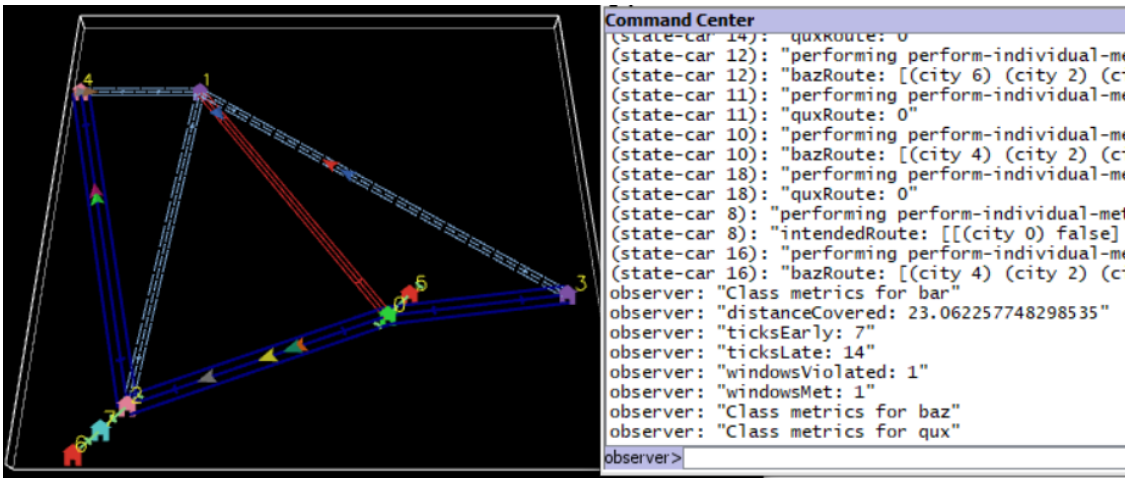


Figure 3.8: Visual representation of the NetLogo simulation generated from the second example.

3.4.2.4 An example result

Figure 3.8 presents a screenshot in which the changes applied in the second example can be seen. The first thing to notice is that the simulation now encompasses a larger number of agents. The edges now appear in different colours and drawing styles that correspond to the applied appearance specifications. The right side of the picture shows the console output that ensues from the definition of the metrics in the metric section. The lower part shows, that the fleet of `bar` agents had to wait an accumulated seven ticks so far serving one customer in time and missing one time window due to the unexpected occurrence of traffic on the streets which at the point in time the optimal routes were calculated was not present.

A close look at the format of the routes of the `baz` agents shows that these routes merely consist of a sequence of cities. For state-car 8, the format is a little different: here, the routes consist of a city together with a boolean value. This is required since Athos needs to insert stopovers between two customers of a tour that are not directly connected by an edge. The boolean value then is used so that `bar` agents can tell a simple stopover node from an actual customer they need to service[15].

[15]Note that for the `qux` agent the route is printed as a 0 value. Metrics need to be implemented for all `AgentBehaviourDescriptions` in the generator and updates to the language necessitate the update of the transformations, which sometimes leads to broken output. See Section 4.1.2, Section 5.6, and Section 5.7, and also Section 8.3.

3.5 General architecture and usage of Athos

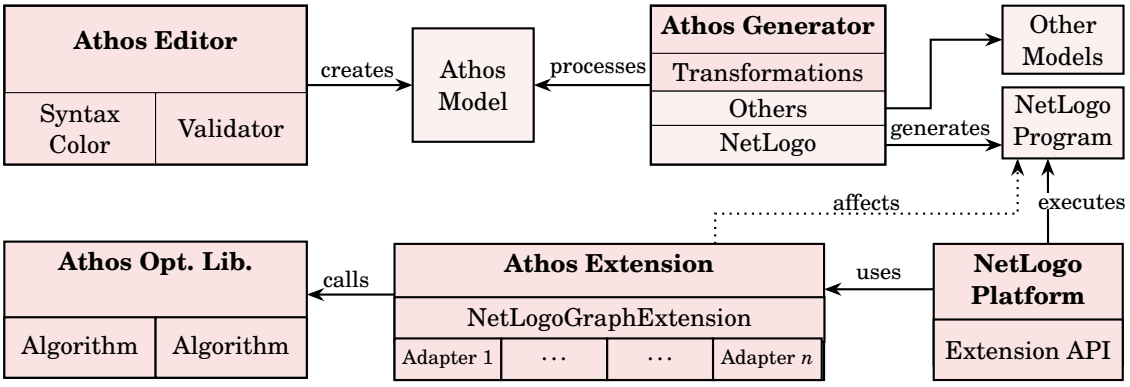


Figure 3.9: Architecture of the Athos development environment.

Figure 3.9 provides an overview on the Athos architecture and how it is used. Athos was implemented using the Xtext[16] language development framework. The reason why Xtext was chosen as the language workbench for this research project was the positive experience the author of this thesis made using the framework in different projects throughout his Bachelor's and Masters' studies. In the conception phase for this project, the author was certain that Xtext provided every feature necessary for the implementation of Athos. This experience with the Xtext framework and additional experience with some of its key components (e.g. the Eclipse modeling framework (EMF) and ANTLR (Parr, 2011)) turned the balance towards the usage of Xtext without detailed consideration and comparison of alternative approaches like Spoofax[17] A formal comparison of the capabilities offered by different language workbenches can be found in (Johnsson and Olsson, 2016). With the capabilities of the Xtext framework, Athos is implemented as an Eclipse plug-in that integrates a dedicated Athos editor into the Eclipse environment. This editor features functionalities such as syntax highlighting, code completion, outline view etc. It also has an integrated validator that checks that no constraints of the language's static semantics are violated.

[16] https://www.eclipse.org/Xtext/
[17] https://www.spoofax.dev/

With the editor, Athos models can be created. The Athos model in the editor is internally stored as an instance of Athos' abstract syntax encoded in an EMF (Steinberg, 2009) model. This EMF model is processed by the generator (see Chapter 5) that transforms the Athos models into executable NetLogo simulations (other platforms can be supported as well which has been shown by means of a a proof of concept (Hoffmann *et al.*, 2018b)). The generated NetLogo code is run on the NetLogo target platform. This platform also offers an extension API. Via this API, the Athos optimisation library is integrated into the illustrated architecture. However, as will be explained in Section 5.10.1, the classes created for the NetLogo extension API do not run the code directly but rely on several adapter classes so the actual algorithms of the Athos optimisation library are decoupled from any target platform.

Fourth Chapter

The syntax and semantics of Athos

This section focuses the syntactical elements of which Athos is comprised together with their respective semantics and notation. The abstract and concrete syntax as well as the static and dynamic semantics are the constituting elements of every (domain-specific) computer language (cf., e.g. Vangheluwe *et al.*, 2007; Cuadrado *et al.*, 2013; Völter and Benz, 2013). Section 4.1 starts out by presenting an overview on the *abstract syntax* of Athos: the elements and their relations are presented together with their respective semantics. Section 4.2 discusses some interesting constraints each Athos model is subjected to in order to be considered valid[1].

4.1 The abstract syntax

4.1.1 Network related meta-model elements

Figure 4.1 provides an overview of those Athos meta-model elements that are related to the network and the functions required to determine movement speeds within the network. Both the `Function` and the `Network` element are direct children of the `Model` root element. The `Network` comprises `Nodes` and `Edges`. `Edges` can be composed to `EdgeGroups`. `Edges` are specialised by `FunctionalEdges` which have

[1] At this point it should be noted that the abstract syntax of a language together with the semantical constraints or the *static semantics* of a language form its *meta-model*. However, in this thesis the terms 'meta model' and 'abstract syntax' are used interchangeably.

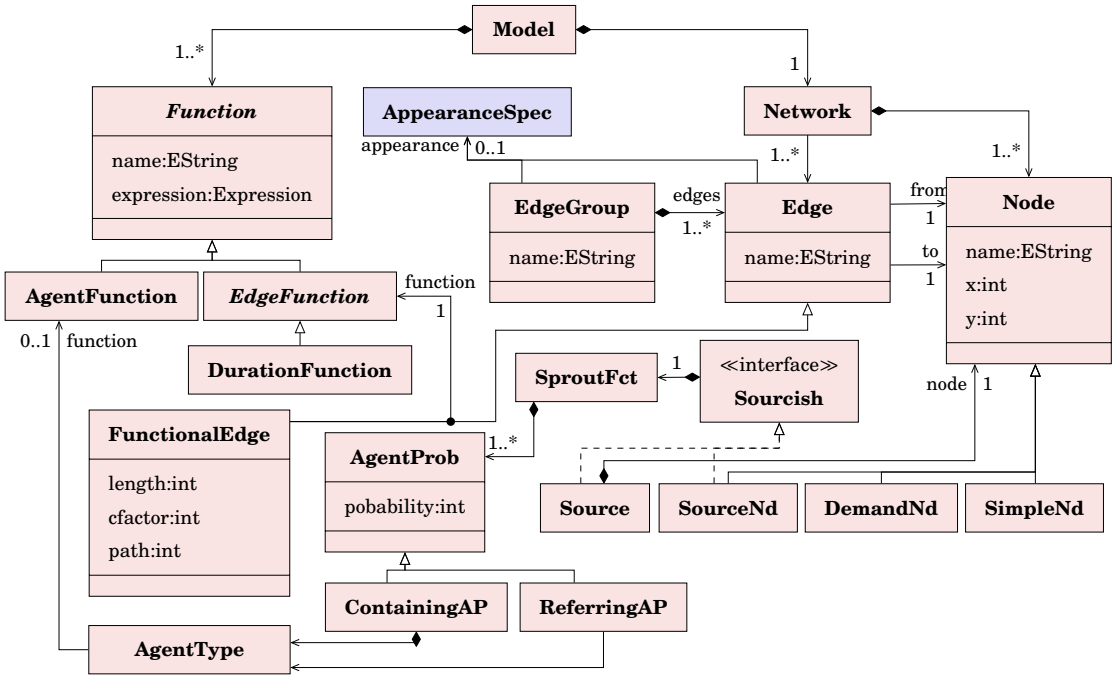


Figure 4.1: The network-related abstract syntax elements of Athos and their relations.

an associated `EdgeFunction`. These functions are used to determine the movement speed of agents on the respective edge. Each edge has exactly one `from` and one `to` Node.

In Athos, there are three different kinds of nodes: `SourceNd` nodes that sprout agents into the network, `DemandNodes` that represent customers in the network. And `SimpleNodes` that mostly serve for navigational purposes so that the geometry of certain streets can be modelled with corners or even bends (through usage of a multitude of nodes that are connected by edges). Nodes that sprout agents into the network implement the `Sourcish` interface which establishes the relation to a `SproutFunction` (also see Section 4.3.1). Functions that sprout agents into the network are not directly associated with an Agent or an `AgentType`. Instead they are composed of a set of `AgentProbabilites` which build the connection to the `AgentType`. This way it is possible to model probabilistic sprout functions.

In order for Athos to allow two different types of specifications of Agents to be sprouted, there are `ContainingAgentProbabilities` and `ReferencingAgentProbabilities`. The former are used for an in-place specification of an `AgentType` to be sprouted, the latter are used to refer to an `AgentType` defined in the agent section of

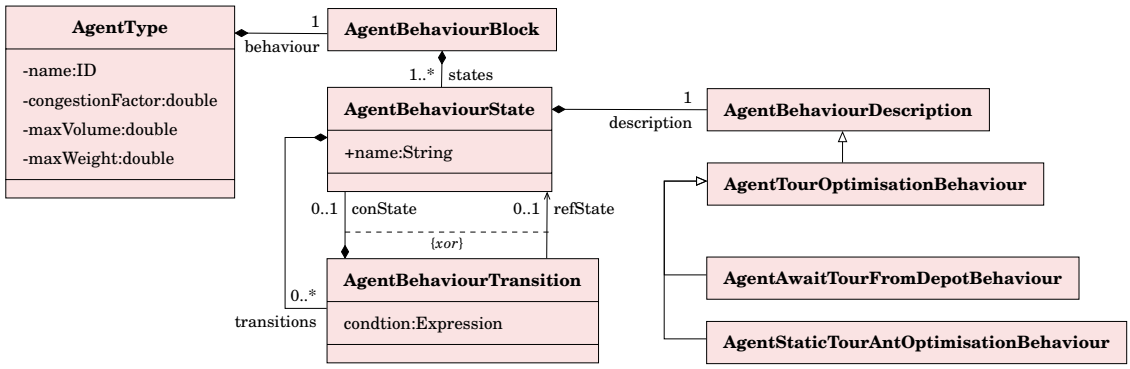


Figure 4.2: Excerpt of the meta-model of Athos depicting elements used to represent agent behaviour (based on (Hoffmann *et al.*, 2019b)).

an Athos model. This is the kind of reference that was used in the presented example programs in Section 3.4.1 and Section 3.4.2. This mechanism is further explained in Section 4.1.3. Athos programs also allow the specification of a special *source section*. In this section, sources are defined by providing a reference to a node from the network section together with the definition of a (probabilistic) sprout function. This way a simple navigation node can become a node capable of sprouting agents. In the last few years, Athos has been under constant development. As such, this feature is mainly kept for reasons of backwards compatibility to prior states of the language thatsince early version of Athos only used this source section mechanism (cf. (Hoffmann *et al.*, 2018b)).

4.1.2 Elements related to agent types and agent behaviour

RQ 1.1 asked for the elements required to model agents together with the behaviour they exhibit during the simulation of vehicle-routing scenarios. As is depicted in Figure 4.2, the meta-model of Athos defines a set of five meta-model classes (with various subclasses) that represent agent behaviour. In Athos, every `AgentType` is associated with exactly one `AgentBehaviourBlock` that comprises an arbitrary number of `AgentBehaviourState`s. These behaviour states are similar to the states found in deterministic finite automatons in that there is an initial state and a set of transitions that allow an agent to change its current state.

Table 4.1: Summary of network-related meta-model elements of Athos together with the keyword initiating the instantiation of the element in an Athos model and a description of the semantics for each element. Abstract elements cannot be instantiated and thus have no associated keyword.

Element	Keyword	Semantics
Model	model	Root element of every Athos model. Also contains information on spatial boundaries.
Edge	vintage	A directed or undirected connection between to nodes. Actually treated as an abstract class but instantiable for reasons of backwards compatibility.
Function	–	Abstract element. Represents most functions in Athos models that assign numerical values to other elements based on the current state of the simulation.
EdgeFunction	–	Abstract element. Represents functions that determine the amount of time in which or the speed by which agents can traverse edges.
DurationFunction	durationFunction	A function which expects an expression that determines the *amount of time* an agent requires to traverse an edge associated with the function.
SpeedFunction	speedFunction	A function which expects an expression that determines the speed (measured in units per tick) by which an agent can traverse an edge associated with the function.
Network	network	The graph-structure comprised of nodes and edges on which the agents/vehicles operate.
Edge	edge	An element that represents any kind of thoroughfare between two nodes (e.g. streets, highways, etc.).
Node	–	Abstract element. A distinct place of interest inside the network (e.g. a customer, a depot, etc.)
SourceNd	source	A node that generates agents that participate in the simulation.

Each `AgentBehaviourState` is associated with exactly one `AgentBehaviourDescription` that represents the behaviour that the agent exhibits while being in the respective state. Each `AgentBehaviourState` is also associated with a set of `AgentBehaviourTransition`s. These transitions define the stimuli that trigger an agent's state transition together with the respective target state the agent is supposed to transition to.

As an example for an interplay of these elements consider an agent that is in a state in which it serves the customers defined in a tour (behaviour description).

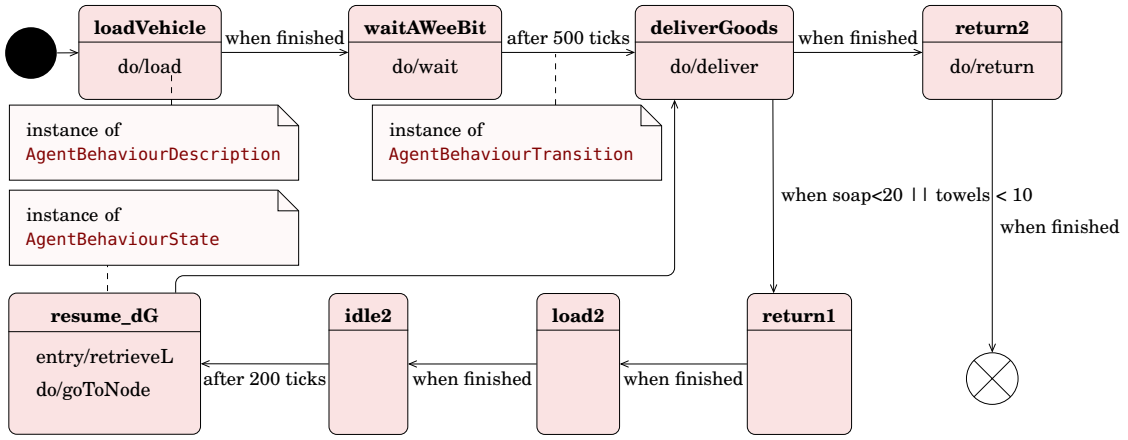


Figure 4.3: State-machine based behaviour of an exemplary delivery agent and its association to the Athos metamodel (adapted from (Hoffmann *et al.*, 2019a)).

In case that the agent's supplies of a given product fall below a certain threshold (condition of a transition), a transition triggers a state-change (state) and the agent returns to a depot and replenishes its stock (behaviour description) and continues in its prior state to serve the remaining customers on the tour (behaviour description). After some time, the simulation clock exceeds a predefined number of temporal units (ticks) which triggers another state change (transition). This time, however, the visible behaviour of the agent does not change (behaviour description) as the agent simply continues to deliver products. However, as the new state of the agent is associated with a different set of transitions, the next time its supplies fall below a given threshold (which might also have changed, condition of transition) the agent does not replenish its stock but returns back to its origin depot where it was sprouted and continues the tour on the next working day (target state of transitions with associated behaviour description).

Figure 4.3 illustrates another example state-machine-based behavioural program for a delivery agent and how it leverages the meta-model elements depicted in Figure 4.2. Each state is an instance of the `AgentBehaviourState` meta-model element. The actual behaviour performed by the agent is represented by the illustrated entry and state behaviours which in Athos are instances of the `AgentBehaviourDescription` element. The transitions between states are instances of the `AgentBehaviourTransition` element.

Listing 4.1: In-place vs. referencing agent type definition.

```
1  ...
2  n0 at (-3, -4)
3        isDepot unit sprouts
4              agentType inPlaceAgentType maxWeight 200
5                behaviour awt awaitTour when finished do die
6                behaviour die vanish
7        customers n3, n4 at 0 latestTime 800
8  n1 at (-6, 8)
9        isDepot unit sprouts referencedAgentType customers n3, n4
10 ...
11 agentTypes
12  agentType referencedAgentType maxWeight 200
13     behaviour awt awaitTour when finished do die
14     behaviour die vanish
```

The agent program modelled in Figure 4.3 defines an agent that starts with loading its cargo space before transitioning into an idling state. After exactly 500 ticks, the agent starts its tour and delivers the loaded goods. In case that the number of available units of the product 'soap' drops below 20 or the available units of 'towels' fall below 10, the agent transitions into a `Return` state[2] that has the agent returning to the nearest depot where it replenishes its stocks before it retrieves the next location to go to and resumes the delivery state and behaviour. When the agent has serviced all customers on its tour, it transitions into a state that has it returning to its destination depot where it leaves the simulation.

4.1.3 In-place agent type specifications

At this point it is important to mention that the presented meta-model currently allows in-place definitions of agent-types. This means that agent types can be defined in the agent type section or even directly inside a sprout function. As is depicted in Section 4.1, a `SproutFct` (sprout function) contains an `AgentProb` which can be either a `ContainingAP` or a `ReferringAP`. The former *contains* an `AgentType` wheras the latter *references* an `AgentType`. Thus, the `ContainingAP` is used for in-place agent type definitions and the `ReferringAP` is used to reference an agent type that is defined in the agent type section in an Athos model.

Both possibilities are exemplified in Listing 4.1. The depot defined in lines 2 – 7 uses a sprout function with an in-place agent definition. The depot defined in

[2]The entry and state behaviours where omitted in the illustration for reasons of brevity.

lines 8 – 9 uses a sprout function that references an agent type defined in the agent types section. This example shows that the referencing approach allows for a better separation of concerns, while the in-place approach mingles the definition of the network structure with the definition of the agent population.

Another drawback of the in-place agent definition is that it considerably increases the complexity of the underlying grammar. This results in situations where additions or changes to the grammar lead to ambiguous grammars in which the parser has multiple alternatives to parse certain parts of an Athos model. These situations are often difficult to retrace even with the support of parsing generator tools. The reason why Athos still allows both ways of defining the type of agents to be created is that it might be more convenient for some users to define an agent type at the exact position in the code where it is needed. Just as Java leaves it up to the user to use either named classes or anonymous classes at different places in the code, Athos was intended to grant users some degrees of freedom so that they can design the structure of their models in a way they deem most appropriate. However, should further empirical studies show that this flexibility does not meet the approval of the language users or even lead to confusion, the in-place agent type definitions will be discarded in the future. Finding out whether users demand for the discussed flexibility or not is part of the future work (see Section 8.3.6.2).

4.2 The static semantics

The success of any DSL is – partially or entirely – determined by the amount of acceptance it gains among its targeted user group. One key prerequisite for a DSL to find wide acceptance is that it supports users in the creation of valid models (Johanson and Hasselbring, 2017, p. 2210). One of the major advantages DSLs have over application libraries for GPLs, is that they can be enriched with constraints that notify end-users whenever they specify models that violate certain assumptions under which the modelling language operates (Dwarakanath *et al.*, 2017, p. 464). Therefore, Athos can potentially support and guide its users in the creation of valid

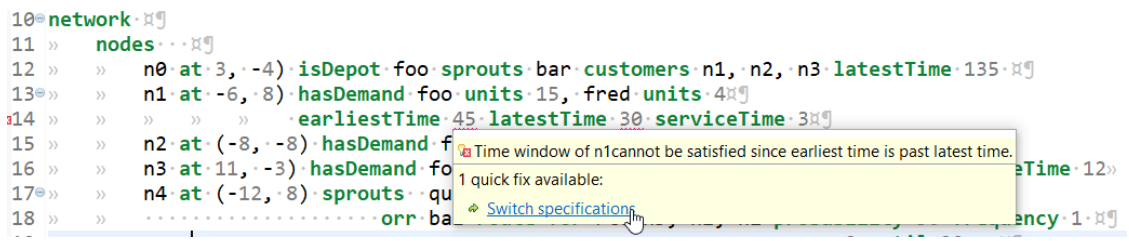


Figure 4.4: Example constraint violation in the editor. The affected parts of the model code are correctly highlighted and a sensible explanation on the violated constraint is given to the user together with an appropriate quick fix.

and executable models in ways that a GPL-based application library (e.g. JSprit[3]) cannot achieve.

These constraints form the static semantics of a language. For DSLs like Athos, they are implemented in a validator that continuously observes the abstract representation of the model (the AST) in the editor and validates it against its set of constraints. If a constraint is violated, the validator can render the model invalid and thus prevent it from being processed by the generator. In addition, the modeller is informed on the respective issue at compile time. Frameworks like Xtext[4] (which was applied for the implementation of Athos) even allow the implementation of automated repair mechanisms known as *quick fixes*.

Figure 4.4 displays an almost trivial but all the more important example: in a moment of inattention a modeller swapped the values for the opening and the closing of the time window in which a customer expects to be serviced. An application library for a GPL cannot detect this mistake and issue a warning about it at compile time. Only at runtime will the library be able to recognise the inconsistency and inform the user by means of an appropriate exception. As can be seen in the screenshot, a DSL like Athos can detect the inconsistency at compile time and thus prevent execution of an inconsistent model. The editor can not only inform the modeller on the problem but highlight the exact places in the textual model that are inconsistent. It can even offer one or several automated repair mechanisms that the user only needs to click on to fix the problem.

[3] https://jsprit.github.io/
[4] https://www.eclipse.org/Xtext/

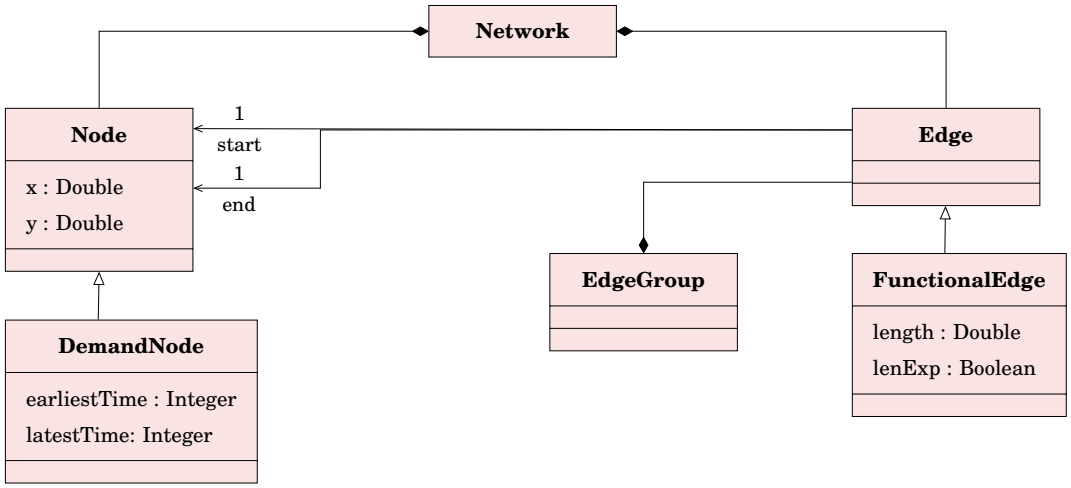
The next two sections present some examples of additional constraints implemented in the validator of the Athos language. Section 4.2.1 presents some simple examples of constraints related to the network section of an Athos model and Section 4.2.2 discusses an example related to the (delivery) behaviour definable in an Athos model. Though the Athos validator is written in the Xtend[5] language, the constraints in the next sections are presented in OCL which is the de-facto standard language in MDSD for the definition of model constraints (Cabot and Gogolla, 2012b). This way, the formal specifications should address a more general audience than the corresponding implementation in Xtend. However, Section 4.2.2 gives an illustrative example of how the constraint implementation in Xtend corresponds to the specification in OCL.

Please note that the sole purpose of the presented examples is the definition of compulsory constraints that an Athos model must fulfil in order to be valid and executable. It is also important to note that at this point, the static semantics of Athos is a part of the language that requires further work. Recognising and implementing sensible constraints thus is an important part of future work to be done on the language (see Section 8.3.3). Currently, Athos does not feature language elements and associated mechanisms that would enable Athos users to define additional constraints that the optimal solution for a given VRP has to fulfil. This is also part of work to be done in the future (see Section 8.3.4).

4.2.1 Constraints related to the network

Especially with regard to the definition of the network in an Athos model, there are several constraints that are comparatively simple but nonetheless important (especially for the generator which operates under the assumption that these constraints are fulfilled). Some of these constraints are presented in Figure 4.5. Here, the first constraint specifies that two different nodes must not have the same coordinates in a simulation. The second constraint ensures that there are no negative

[5] https://www.eclipse.org/xtend/




```
context   Node
inv:      Node::allInstances()
            ->forAll(n | n <> self implies not (n.x == self.x and n.y== self.y)

context   FunctionalEdge
inv:      self.length > 0

context   DemandNode
inv:      DemandNode::allInstances()
            ->forAll(n | n <> self implies n.earliestTime < n.latestTime)

context   FunctionalEdge::length : Double
derive:   if not lenExp then
            sqrt(pow(self.start.x - self.end.x,2) + pow(self.start.y - self.end.y,2))
          endif
```

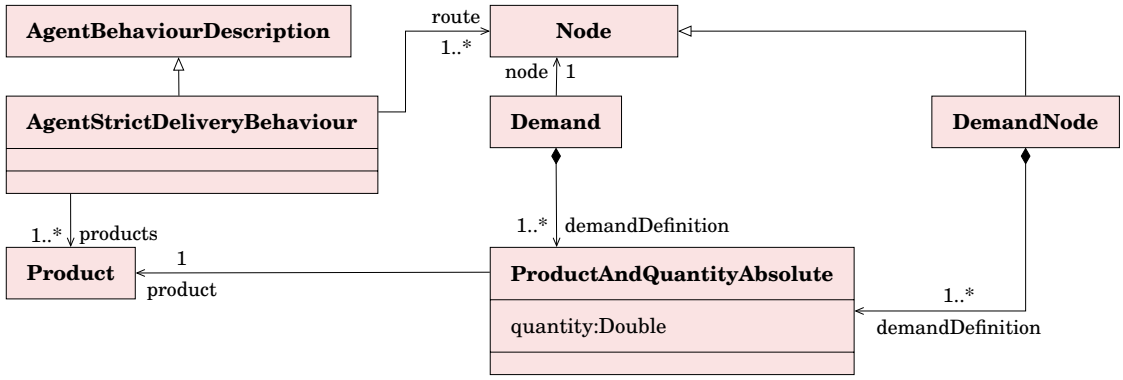
Figure 4.5: Constraints in OCL that (1) ensure that nodes are not linked to themselves, (2) every edge has a positive length, (3) a time window opens before it closes; together with a specification that determines that if no explicit length is specified, the length of an edge is the Euclidean distance of the nodes it connects.

lengths in the definition of edges. The third OCL expression is a formal definition of the example given in the introduction to this section: it states that the time upon which a customer's time window opens must be before the time the time window ends. In other words, the vehicle must be given a period in time of at least one tick in which it may serve the customer.

The last OCL expression in Figure 4.5 is not an actual constraint but the definition of a semantic rule (or assumption) applied by the generator: it states that in case that no length for an edge is explicitly set (as was the case in the presented examples in Section 3.4.1 and Section 3.4.2), the length of the edge is by default the Euclidean distance of the nodes it connects. The codification of this rule as an

OCL expression was done for presentational purposes only. As was already said, the actual implementation of this rule was done in the transformations of the Athos generator,

4.2.2 Constraints related to the agent behaviour




```
context   AgentStrictDeliveryBehaviour
inv:      self.route->forAll(n | self.products
            ->exists(p | n.demandDefinition->map(product)->contains(p)))
          and
          Demand.allInstances().select(d | self.route->contains(d.node))
            ->forAll(d | self.products
            ->exists(p | d.demandDefinition->map(product)->contains(p)))
```

Figure 4.6: Constraint which ensures that customers have a demand for the good delivered by the agent.

As Listing 3.1 and Listing 3.2 in Section 3.4 have shown, one of the most common modelling tasks in Athos is the assignment of customers to depots. In complex simulations with large numbers of agents and depots, a common mistake is the assignment of customers to the wrong depot. Through the definition of constraints that form Athos' static semantics, Athos can prevent such modelling mistakes through its validator. Figure 4.6 gives an example for such a constraint. It shows how the Athos validator ensures that a customer that is referred to in the `route` list of an agent is of type `DemandNode` (which is a necessary condition). Moreover, it is also ensured that the customer actually has a demand for the product that the respective agent intends to deliver.

As was mentioned at the beginning of this section, OCL is the quasi standard language for the definition of additional constraints for (UML) models – especially

Listing 4.2: Constraint that ensures that customers demand for delivered product.

```
@Check
def checkForUnmatchingProducts(AgentStrictDeliveryBehaviour asdb){
  if(!asdb.everything){
    for (Node n : asdb.route.filter(DemandNode) ){
      if (!asdb.products
          .exists[p | (n as DemandNode).demandDefinition.map[product].contains(p)])
        warning('''Node «n.name» has no demand for any of the delivered products.''',
             asdb,Athos5Package.eINSTANCE.agentRoutingBehaviour_Route,
             asdb.route.indexOf(n), DELIVERY_TO_NODE_WITHOUT_DEMAND
        )
    }
    val model = asdb.getContainerOfType(Model)
    for (Node n: asdb.route){
      val demandForNode = model.network.demands.filter[node == n]
      if(!demandForNode.nullOrEmpty &&
          !asdb.products
            .exists[p | demandForNode.map[demandDefinition]
            .flatten.map[product].toSet.contains(p)]
      )
        warning('''Node «n.name» has no demand for any of the delivered products.''',
             asdb,Athos5Package.eINSTANCE.agentRoutingBehaviour_Route,
             asdb.route.indexOf(n), DELIVERY_TO_NODE_WITHOUT_DEMAND
        )
    }
  }
}
```

for those models applied in an MDSD process. The introduction also mentioned that for the Athos validator, the actual implementation of all constraints was done with the Xtendhttps://www.eclipse.org/xtend/ programming language. In order to convey an impression of the actual implementation, Listing 4.2 shows how the OCL constraint from Figure 4.6 is implemented in the Athos validator.

4.2.3 Overview on currently active constraints

Table 4.2 summarises the constraints that are currently active in the Athos validator. The first column of the table gives the name of the constraint (or more precisely: the *issue code* that is raised upon violation of the respective constraint). The second column states whether an *error* or a *warning* is issued when a model violates the respective constraint. When an error is raised, the generator will not process the model and thus the model cannot be executed. Errors are raised for constraint violations that will definitely lead to breakdowns at runtime. There are also cases in which a modelling decision is not guaranteed but highly likely to cause problems if modellers do not exactly know what they are doing. For these kind of issues,

Table 4.2: Summary of currently active constraints in the Athos validator. Errors prevent a model from execution; warnings inform on model elements that are likely to cause problems at runtime.

Constraint	Type	Model section	Description
AMBIGOUS_ NODE_ COORDINATES	Error	Network	Ensures that two different nodes do not share the same coordinates. This constraint might be dropped in later versions; currently, the generated code cannot handle different nodes with identical coordinates.
NEGATIVE_ EDGE_LENGTH	Error	Network	Ensures that all edges have a positive length. If the length of an edge is set to zero, the Euclidean distance of the nodes connected by the edge is assumed to be the edge's length.
UNDEFINED_ DISTRIBUTION	Error	Network	Ensures that for each agent type in a probability distribution the model contains a value greater than zero.
WRONGLY_ WEIGHTED_ DISTRIBUTION	Error	Network	Ensures that the probability values of a probability distribution add up to 100.
UNNECESSARY_ PROBABILITY_ DEFINED	Warning	Network	Informs the modeller that a probability distribution with only one vehicle type should be replaced by a simple agent type specification (an agent of this type is then guaranteed to be created).
DELIVERY_ TO_NODE_ WITHOUT_ DEMAND	Warning	Behaviour	Warns the modeller that the respective node appears in the list of customers of an agent even though it does not have a demand specified for any of the products delivered by the agent.
DELIVERY_TO_ NAVIGATION_ NODE	Warning	Behaviour	Warns the modeller that the respective node appears in the list of customers of an agent, even though it is not a customer node (i.e. no demand was defined for the node).
WRONG_ PARAMETER_ NAME_STATED	Warning	Network (External)	Used for Athos' extension mechanism to ensure that certain agents which an external algorithm assumes to be created by a source, the network defines a corresponding sprouting function (cf. (Hoffmann *et al.*, 2020)).

warnings are raised that inform modellers on modelling decisions that require extra attention. The last column describes the intention (semantics) of the respective constraint.

As can be seen from the table, the number of currently active constraints is somewhat limited. This is because of the limited experience with Athos in practical

application. In order to define and implement sensible constraints, i.e. constraints that prevent modelling mistakes which many users are inclined to make, it is crucial to receive feedback from actual language users. Therefore, the definition and implementation of more constraints (see Section 8.3.3) together with application of Athos by practitioners (see Section 8.3.4) will be part of future work to be done.

4.3 The concrete syntax

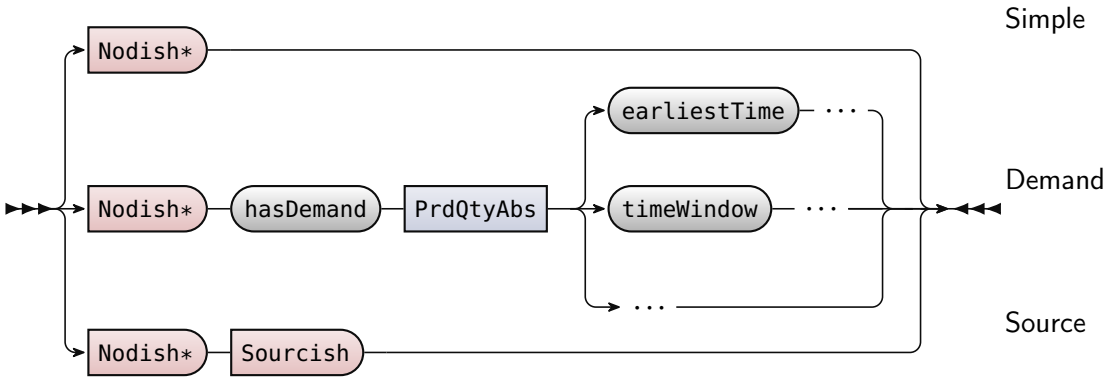
This section discusses some parts of the concrete syntax of Athos. This is done in order to provide some insight on how the language actually looks and feels when used to model traffic simulation and transport optimisation scenarios. For reasons of brevity, only a selection of the most interesting and most important parts of the concrete syntax are presented and discussed. The complete definition of the concrete syntax in Xtext's EBNF-based notation is provided in Appendix A.

4.3.1 Definition of nodes

Figure 4.7 illustrates the concrete syntax of Athos for the definition of the three different types of nodes. The illustration shows both a graphical 'rail road diagram' and also the textual extended Backus-Naur form (EBNF) dialect used in the Xtext framework. A special characteristic of the Xtext-EBNF notation is that it also requires the specification of attributes in which the respective elements are stored (in Figure 4.7, `demandDefinition`, `product`, and `quantity` are such attributes). This is because the framework derives the meta-model of the language from these grammar rules, i.e. all meta-model `classes` together with their `attributes` and `references`. The derived meta-model is implemented by means of the EMF (Steinberg, 2009). The official website discusses the details of how the meta-model is derived from the grammar rules at great length[6]. For this section, the following, somewhat oversimplified explanation is sufficient: grammar rules are translated into `EClasses` with the specified `EAttributes`. References to other EClasses are specified by using

[6] `https://www.eclipse.org/Xtext/documentation/301_grammarlanguage.html`

Node:




```
Node: SimpleNode | DemandNode | SourceNode;

SimpleNode: Nodish;

DemandNode: Nodish 'hasDemand' demandDefinition=PrdQtyAbs
            ('earliestTime' ... | 'timeWindow' ... | ... )

SourceNode: Nodish Sourcish;
```

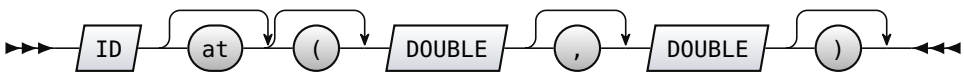
PrdQtyAbs:


```
PrdQtyAbs:  product=[Product] 'units' quantity=Double
```

Figure 4.7: Concrete syntax for the definition of simple nodes, demand nodes and source nodes and the specification of absolute product quantities.

the name of another grammar rule in square brackets (e.g. [Product] in Figure 4.7 is such a reference). The usage of a string within curly braces results in the generation of another `EClass` that has the `EClass` generated for the rule in which the curly braces have been used as a superclass see Figure 4.10.

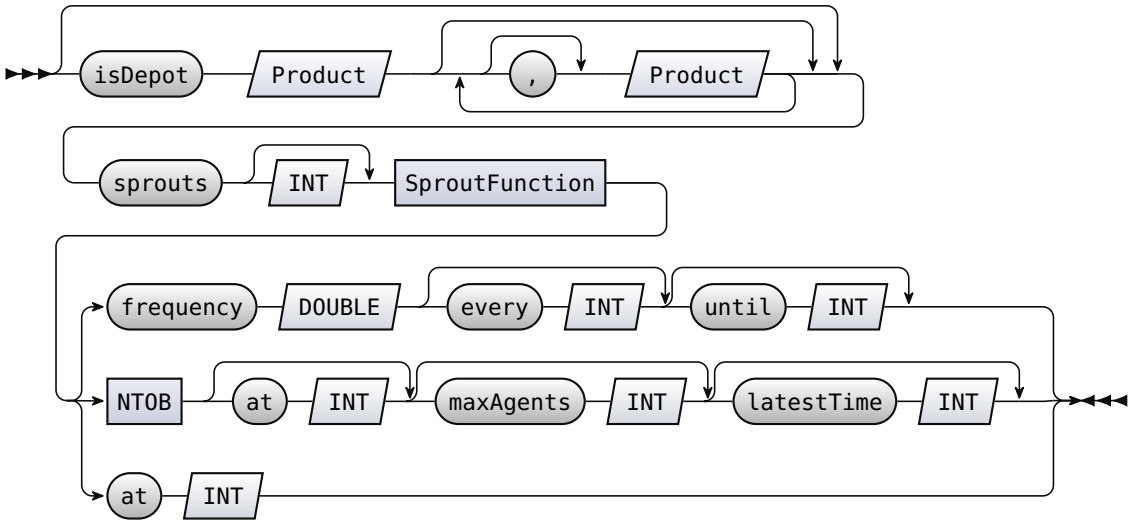
Nodish*:




```
Nodish: name=ID 'at'? '(' x=DOUBLE ','? y=DOUBLE ')'?
```

Figure 4.8: Concrete syntax for the definition of a `Nodish` element.

Sourcish:




```
Sourcish:
  (isDepot?='isDepot' products+=[Product] (','? products+=[Product])*)?
  'sprouts' factor=INT? sproutFunction=SproutFunction
      ( 'frequency' frequency=DOUBLE ('every' every=INT)? ('until' until=INT)  |
        tourOptimisation=NTOB ('at' at=INT)?
            ('maxAgents' maxAgentsINT)? ('latestTime' latestTime=INT)?    |
        simpleStart?='at' at=INT )
```

Figure 4.9: Syntax diagram for the Sourcish rule fragment that is used for nodes from which agents are generated. For the meta-model of the language, rules that use the Sourcish fragment are translated into EClasses that inherit from the class Sourcish.

As was already mentioned, there are three different types of nodes in Athos (see Section 4.1.1). The concrete syntax for the definition of all three node types starts with a *rule fragment* (a reusable container for syntactical structures (see, e.g., (Bettini, 2016, pp. 208–209))) named Nodish. The grammar stored in this fragment is shown in Figure 4.8. As can be seen, the specification of all three node types begins with the definition of a mandatory identifier followed by the coordinates of the node. Optionally, brackets and a comma can be used for the specification of coordinates[7]. For the definition of a SimpleNode the syntactical elements specified

[7]A closer look at this rules reveals that it allows the usage of a closing bracket without a corresponding opening bracket. Alternatively, two paths could be defined, one that mandates the usage of both brackets and one that does without any brackets. On the other hand, there is no harm in allowing users to specify coordinates with only closing brackets (see Figure 4.10).

in the Nodish rule fragment are sufficient because a simple node requires nothing but an identifier and a pair of coordinates.

The characteristic element in the specification of a DemandNode is the keyword (or terminal rule in EBNF terminology) hasDemand. Subsequent to this keyword, a reference to a product specified in the product section together with a quantity (double value) must be provided. This is expressed via the call to the non-terminal rule PrdQtyAbs[8] (product and quantity absolute)[9]. The next element is an optional definition of a time window. This time window can be defined in two syntactically different (yet semantically equivalent) ways: A longer version that explicitly uses the keywords earliestTime and latestTime each followed by a double value that specifies the respective tick for the opening and closing of the time window. Alternatively, a more concise version that uses the keyword timeWindow followed by two double values (optionally separated by a comma) can be used. Future versions of Athos might opt to deprecate and remove one of the two ways (see Section 8.3.6.2).

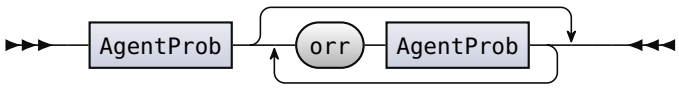
The definition of a SourceNode introduces yet another rule fragment. The Sourcish rule fragment is used to specify depots for a set of products as well as for the specification of nodes from which agents are generated (or 'sprouted', in NetLogo terms) into the simulation. As can be seen, in contrast to the Nodish rule fragment, there is no asterisk used in the Sourcish rule fragment. The asterisk notation is borrowed from the Xtext framework (see, e.g., (Bettini, 2016, pp. 208–209)): the Nodish fragment is merely a reference to the respective syntax elements and was only defined for the purpose of reusability. By contrast, in the meta-model of the language the Sourcish fragment will become the super class of every class derived from a grammar rule that uses this fragment (see Section 4.1.1). In other words, in the Athos meta model a SourceNode is a specialisation of a Sourcish element (see Figure 4.1 in Section 4.1.1).

[8]The grammar also supports the definition of relative quantities. Relative quantities are, however, currently not supported by the generator. They are part of the language because in future versions it will be possible to define that a delivery agent is to deliver 50 per cent of its cargo to customer A and 25 per cent each to customers B and C.

[9]The name was abbreviated so as to fit it into the graphic. In the complete grammar presented in Appendix A the long form ProductAndQuantityAbsolute is used.

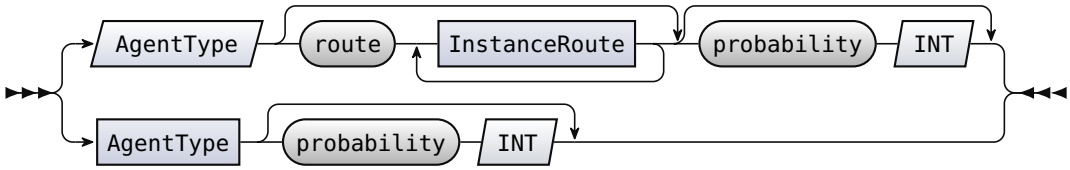
The concrete syntax subsumed under the Sourcish fragment is illustrated in Figure 4.9. As can be seen in the railroad diagram, it begins with the terminal rule isDepot. This defines the isDepot keyword that is used for the definition of depots (see node n0 in Line 10 shown in the introductory example in Listing 3.1). The Sourcish rule defines the specification of the isDepot keyword as optional. However, if this keyword is used, it mandates the specification of at least one reference to a product defined in the product section. After that the mandatory sprouts keyword is to be used followed by an integer value. The meaning of this integer is that it defines how many agents are to be sprouted into the network.

SproutFunction:




```
SproutFunction:
  agentProbabilities+=AgentProb ('orr' agentProbabilities+=AgentProb)*
```

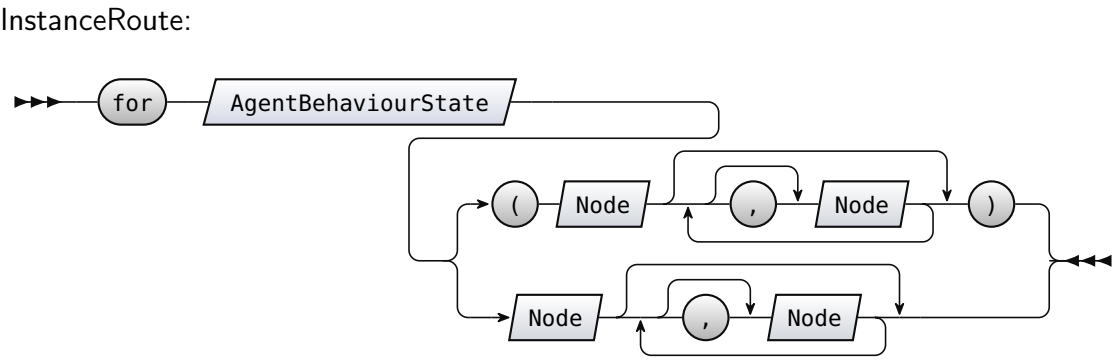
AgentProb:




```
AgentProb:
  {ReferringAgentProb}
    (agentReference=[AgentType]
      (instanceRoutes?='route' setOfInstanceRoutes+=InstanceRoute+)?
        ('probability' probability=INT)?)
  |
  {EncapsulatingAgentProb}
    (agentContainment=AgentType ('probability' probability=INT)?)
```

Figure 4.10: Concrete syntax for the definition of SproutFunctions and AgentProbs in Athos.

The SproutFunction rule that is referenced next is illustrated in Figure 4.10. A SproutFunction allows the definition of and/or reference to one or more AgentTypes




```
InstanceRoute: 'for' state=[AgentBehaviourState]
               ( '(' route+=[Node] (','? route+=[Node])* ')'     |
                     route+=[Node] (','? route+=[Node])*  )
```

Figure 4.11: Concrete syntax for the definition of an individual route for an AgentType sprouted at a source.

(also see Section 4.1.3) separated by the keyword orr[10] that are sprouted with a given probability[11].

As can be seen from the upper railroad in Figure 4.10, for cross-referenced AgentTypes, it is possible to define a route for the agents to be sprouted individually for the respective source. This is referred to as an InstanceRoute. This way, different sources can sprout the same AgentType (in other word reuse the AgentType) with different routes. This was shown in Listing 3.2 of Section 3.4.2 where nodes n4 and n5 both sprout the same baz agent but assign different routes to it. The concrete syntax for the definition of such an InstanceRoute is illustrated in Figure 4.11. It requires the keyword for followed by a reference to an AgentBehaviourState which currently must be the first AgentBehaviourState defined for an agent[12]. After that, a sequence of Nodes that form the route is to be specified.

[10]The spelling with a double r is necessary to distinguish this orr from the or used in logical expressions that can be used in AgentBehaviourTransitions.

[11]The static semantics of Athos defines a constraint that ensures that these probabilities are in the interval [0,100], are not negative, and add up to 100.

[12]The intention here is that an AgentType can comprise many different AgentBehaviourStates with associated AgentBehaviourDescriptions. For a complete reusability of such AgentTypes, the routes for all AgentBehaviourStates (more precisely their associated AgentBehaviourDescriptions) must be definable individually for each source that is to sprout this AgentType.

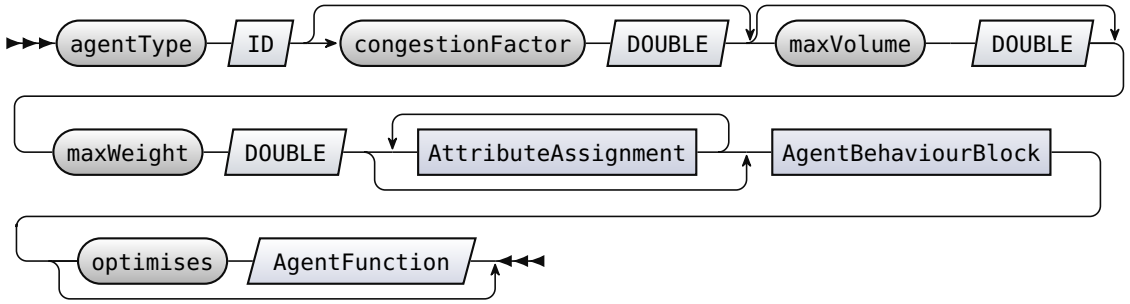
There are different ways in which agents can be sprouted into the network (also see Listing 3.2 of Section 5.2 in this regard). Which one is used is determined by which rail is taken after the definition of the `SproutFunction`. This, is shown in the lower part of Figure 4.9. The keyword `frequency` is used to recognise that the upper rail is to be taken. This path is used to define sources that continuously sprout agents into the network. In Listing 3.2 of Section 3.4.2 this is done by node `n4`. The lower rail is recognised by the keyword `at`. This way one-time sprouts are defined as can be seen with node `n7` in Listing 3.2.

The middle-rail in Figure 4.9 refers to yet another non-terminal rule `NodeTourOptimisationBehaviour` (abbreviated `NTOB`) which is not illustrated here. It is a very simple non-terminal rule that calls one of three different non-terminal rules each of which begins with the keyword `customers` followed by a sequence of nodes (see page 279 in Appendix A). Notice how then the keyword `customers` is used to recognise that the middle rail is to be taken and how the property `tourOptimisation` is set to true. This kind of sprout function defines depots that are associated with a fleet of vehicles that wait for the depot to assign them a tour to service. This mechanism was used for node `n0` in both introductory examples Listing 3.1 in Section 3.4.1 and Listing 3.2 in Section 3.4.2.

4.3.2 Agent type and behaviour related concrete syntax

One of the most important aspects of Athos is that it allows for the definition of agents and their respective behavioural patterns within the simulation. Section 4.1.2 introduced the meta-model elements Athos features for this purpose. It also presented an exemplary use-case in which the meta-model elements were instantiated to model a delivery agent. This section presents the concrete syntax of Athos that is used to specify agent types and their behaviour in Athos models (and thus to actually perform the discussed meta-model instantiations).

AgentType:




```
{AgentType} 'agentType' name=ID ('congestionFactor' congestionFactor=Double)?
('maxVolume' maxVolume=Double)? ('maxWeight' maxWeight=Double)?
(attributeAssignments+=AttributeAssignment)* behaviour=AgentBehaviourBlock
(individualOptimizsation?='optimises' function[AgentFunction])? ;
```

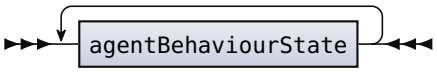
Figure 4.12: Railroad diagram describing the concrete syntax of Athos for the definition of an agent type.

As is depicted in Figure 4.12, the definition of an `AgentType` starts with the `agentType` keyword followed by a name for the agent type that is about to be defined. Subsequently, a congestion factor, a maximum volume and maximum weight can optionally be specified for the agent type. The semantics of these syntactical elements is given in Table 4.3. In addition to these natively supported specifications, additional attributes and their respective values can be defined for the agent type. After that, the `AgentBehaviourBlock` – the centrepiece for behavioural specifications – has to be defined. Finally, the agent type can be assigned an `AgentFunction` which serves as an objective function that the agent seeks to optimise.

Table 4.3: Summary of agent-related syntax elements in Athos and a description of their respective semantics.

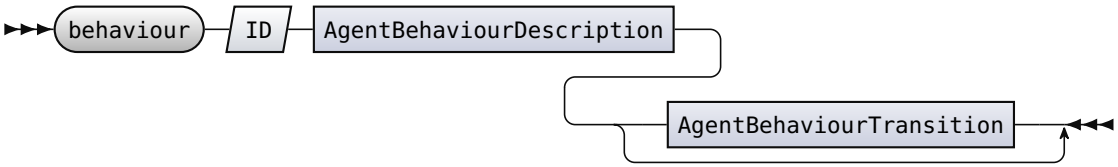
Element	Keyword	Semantics
AgentType	agentType	*[EClass]* – A certain agent type (or agent class) whose instances share a defined set of (built-in) properties and additional attributes (and their values)
Congestion factor	congestionFactor	*[EAttribute]* –: The value of this property represents the extent to which the respective agent type contributes to congesting the current road. Road functions can use a built-in primitive that represents the accumulated congestion factors of all vehicles currently located at the respective road.
Maximal volume	maxVolume	*[EAttribute]* – The maximal loading space the vehicle is able to carry (if specified together with the maxWeight of a vehicle, the more restrictive value will apply).
Maximal weight	maxWeight	*[EAttribute]* – The admissible total weight the vehicle can transport
Attribute assignment		*[Subrule]* – References an attribute declared in the agent attributes section and assigns a specific value or value distribution to this attribute.
Agent function		*[EReference]* – References an agent function specified in the function section of an Athos program. This function is an individual objective function for this type of agent.

AgentBehaviourBlock:



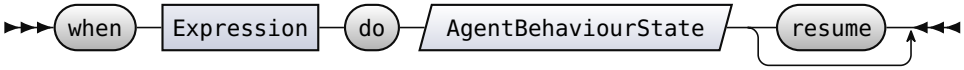

```
{AgentBehaviourBlock} (agentBehaviourStates+=AgentBehaviourState)+;
```

AgentBehaviourState:




```
'behaviour' name=ID description=AgentBehaviourDescription
 (transition+=AgentBehaviourTransition)*;
```

AgentBehaviourTransition:



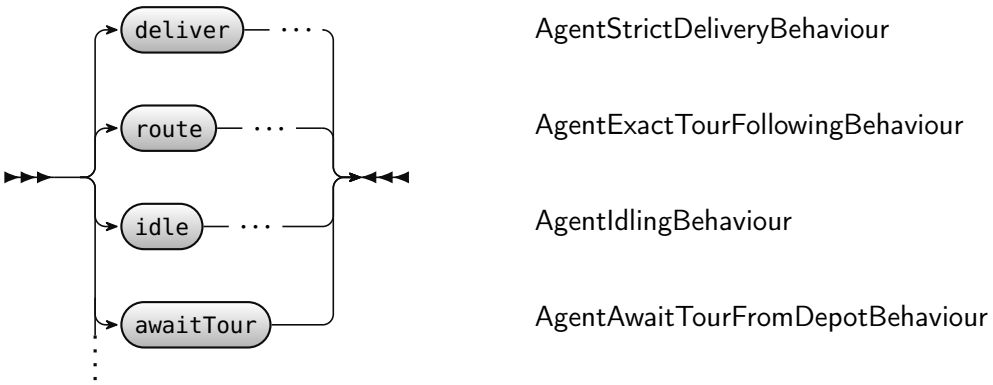

```
'when' condition=Expression 'do' refState=[AgentBehaviourState]
(resume?='resume')?;
```

Figure 4.13: Concrete syntax for the definition of agent behaviour in Athos.

Figure 4.13 shows that the `AgentBehaviourBlock` rule is just a delegation rule without any token consumption. The fact that this non-terminal rule calls the `AgentBehaviourState` rule multiple times (but at least once) results in the `AgentBehaviourBlock` being a container element for a non-empty set of `AgentBehaviourStates`. An `AgentBehaviourState` is introduced with the keyword `behaviour` followed by a name for the respective `AgentBehaviourState`. Next, the actual `AgentBehaviourDescription` – which will be discussed in more detail in a moment– is to be defined before a set of `AgentBehaviourTransitions` concludes the specification of the agent's behaviour.

As can be seen in the third part of Figure 4.13, a transition is specified with the keyword `when` together with an expression that must evaluate to true in order to trigger the respective transition. Subsequent to the transition expression, the keyword `do` precedes a reference to the state that the agent is to transition into. Optionally, the keyword `resume` can be used to indicate that the agent is supposed

to memorise where the target state was previously left and to start from there. As an example, an agent might transition from a state in which it processes a set of customers to a state where it returns to a nearby depot to replenish its stock. After reloading, the agent can return to the delivery state. To indicate that the agent is to continue its tour where it was suspended, the `resume` keyword must be used.



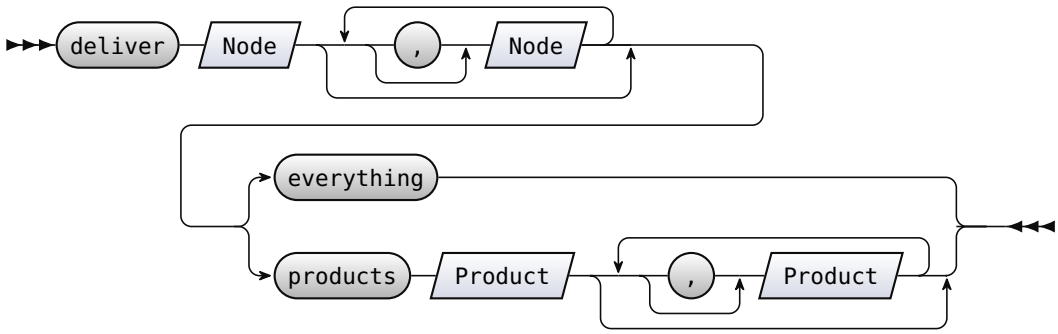

```
AgentBehaviourDescription:
AgentStrictDeliveryBehaviour | AgentExactTourFollowingBehaviour |
AgentIdlingBehaviour | AgentAwaitTourFromDepotBehaviour| ... ;
```

Figure 4.14: Railroad diagram depicting an excerpt of keywords for different `AgentBehaviourDescriptions`.

As was said earlier, the actual behaviour that an agent exhibits is determined by which subclass of `AgentBehaviourDescription` is instantiated and associated with the `AgentBehaviourState`. It was also said that the keyword after the name of the `AgentBehaviourState` indicates which exact `AgentBehaviourDescription` to use. This is depicted in Figure 4.14. For example, the keyword `delivery` indicates that an instance of `AgentStrictDeliveryBehaviour` is to be associated with the `AgentBehaviourState`. This behaviour description has the agent perform a pre-defined set of deliveries to customers in the network. The illustration also gives a few more examples for possible behaviour descriptions, e.g. a behaviour description that has the agent just visit (not service) a set of nodes in a predefined-order (`AgentExactTourFollowingBehaviour` indicated by the keyword `route`), a behaviour description that has the agent do nothing for a given amount of time

(AgentIdlingBehaviour indicated by the keyword idle), or a behaviour description that has the agent wait at a depot until it is assigned a tour by the depot (AgentAwaitTourFromDepotBehaviour indicated by the keyword awaitTour).

AgentStrictDeliveryBehaviour:




```
AgentStrictDeliveryBehaviour:
 'deliver' route+=[Node] (','? route+=[Node])*
     (everythins?='everything' )  |
     ('products'+=[Product] (','? products+=[Product])*)
```

Figure 4.15: Concrete syntax for the specification of an AgentStrictDeliveryBehaviour.

The complete concrete syntax for the specification of an AgentStrictDeliveryBehaviour is shown in Figure 4.15: after the aforementioned deliver keyword, one or more nodes to be serviced must be specified (optionally separated by a comma). The agent can either deliver every product the customer demands (indicated by the keyword everything) or only a certain set of products (indicated by the keyword product).

Fifth Chapter

Transformation of Athos into NetLogo simulations

This chapter discusses the definition of Athos' execution semantics (also dynamic semantics). The execution semantics of Athos are defined by its generator (see (Völter and Benz, 2013, pp. 82–83)) that comprises a set of transformations which map Athos models to executable textual NetLogo models. The Xtext framework that was used to implement Athos refers to these transformations as *template methods*[1], so these terms will be used interchangeably throughout this chapter. The aim of this chapter is to provide a general insight on the implementational details that ensure the correct transformation of descriptive Athos models into procedural agent-based NetLogo simulations. In order to do so, this chapter is bound to present some technical details that require a basic understanding of the functionalities of the Xtext framework. Though the author of this thesis took care to provide the background information necessary for understanding the contents of this chapter, the referenced resources can be helpful to get an even deeper insight into the technicalities of the Xtext framework.

The rest of this chapter is organised as follows: Section 5.1 introduces a mechanism designed to schedule the creation of agents in the NetLogo simulation according to the specifications given in the underlying Athos model. Section 5.2 then continues

[1] https://www.eclipse.org/xtend/documentation/203_xtend_expressions.html#templates

with a discussion of the general flow of control that the generator establishes for every NetLogo simulation. This is not only important for a comprehensive presentation of the mechanics found in every generated NetLogo model, but also provides the necessary context for subsequent sections that present the detailed generation mechanisms. Section 5.3 then presents the internal structure of the Athos generator. It starts out with a general overview which is followed by a detailed presentation of the transformations of the nodes and agents of an Athos model into their respective representational elements in the NetLogo simulation model.

5.1 The command dictionary

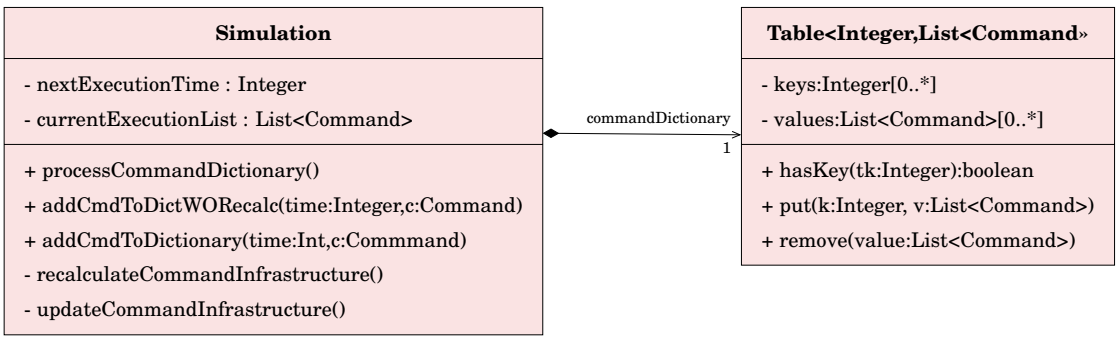


Figure 5.1: The command processing infrastructure of every generated NetLogo simulation.

Every NetLogo simulation created by the Athos generator features two mechanisms by which Agents are *sprouted* into the simulation. For those source nodes for which a continuous or cyclic function was defined in the Athos model, a mechanism based on two distinct functions is applied. These will be discussed in Section 5.2. For depot tours and for agents that are modelled to commence at a specific tick a command processing infrastructure centred around a *command dictionary* data structure is used for the implementation of these agent generations.

This mechanism is illustrated in Figure 5.1. The `commandDictionary` is a table that uses integers as keys and lists (of `Command` instances) as values. The integers represent the exact tick at which the set of commands in the associated list have to be executed. The command processing infrastructure provides a similar execution logic like those found in DES: the global variable `nextExecutionTime` stores the

exact tick at which the next batch of commands must be processed. In contrast to most DES simulations, the generated NetLogo ABM simulation does not discretely jump along these execution times. Instead, every tick is simulated and in each tick the simulation checks whether there are commands to be executed. This polling is done via the `processCommandDictionary()` method[2].

In case that the simulation's tick counter value is below the value stored in the `nextExecutionTime` variable, or in case that there is a negative value in said variable (meaning that the command list is empty), there are no commands to be executed and the simulation can continue to *ask* all agents to perform their next steps according to their current states. In case that the time for the next commands has come, the list of commands is executed. After that the list of executed commands is removed and the next point in time at which commands are to be executed is to be determined together with the actual list of commands to be executed. This is done via the `recalculateCommandInfrastructure()` and `updateCommandInfrastructure()` methods.

5.2 General flow of control

The UML sequence diagram presented in Figure 5.2[3] illustrates the control flow of every generated NetLogo simulation. The user must initiate the `setup` process of the simulation model by pressing the 'setup button' of the generated simulation interface. When the user presses this button, the *observer agent* (the main controller of the simulation) executes the `setup` command which triggers the execution of the `setup-cities` and `setup-links` command. The former instantiates all the *city agents* (which were derived from Athos nodes) whereas the latter instantiates the `DRoads` (directed roads or links) (derived from Athos edges) that connect the cities.

[2]The correct term here would be 'Command' since NetLogo does not provide the concept of 'methods.' However, in this subsection, the term method is used in order to avoid confusion with the `Command`s in the `CommandDictionary`.

[3]Note that the presented diagram is for illustrative purposes only. Though it follows the general syntax and semantics defined for UML sequence diagrams, it is not intended to be in full compliance to any UML 2.x specification.

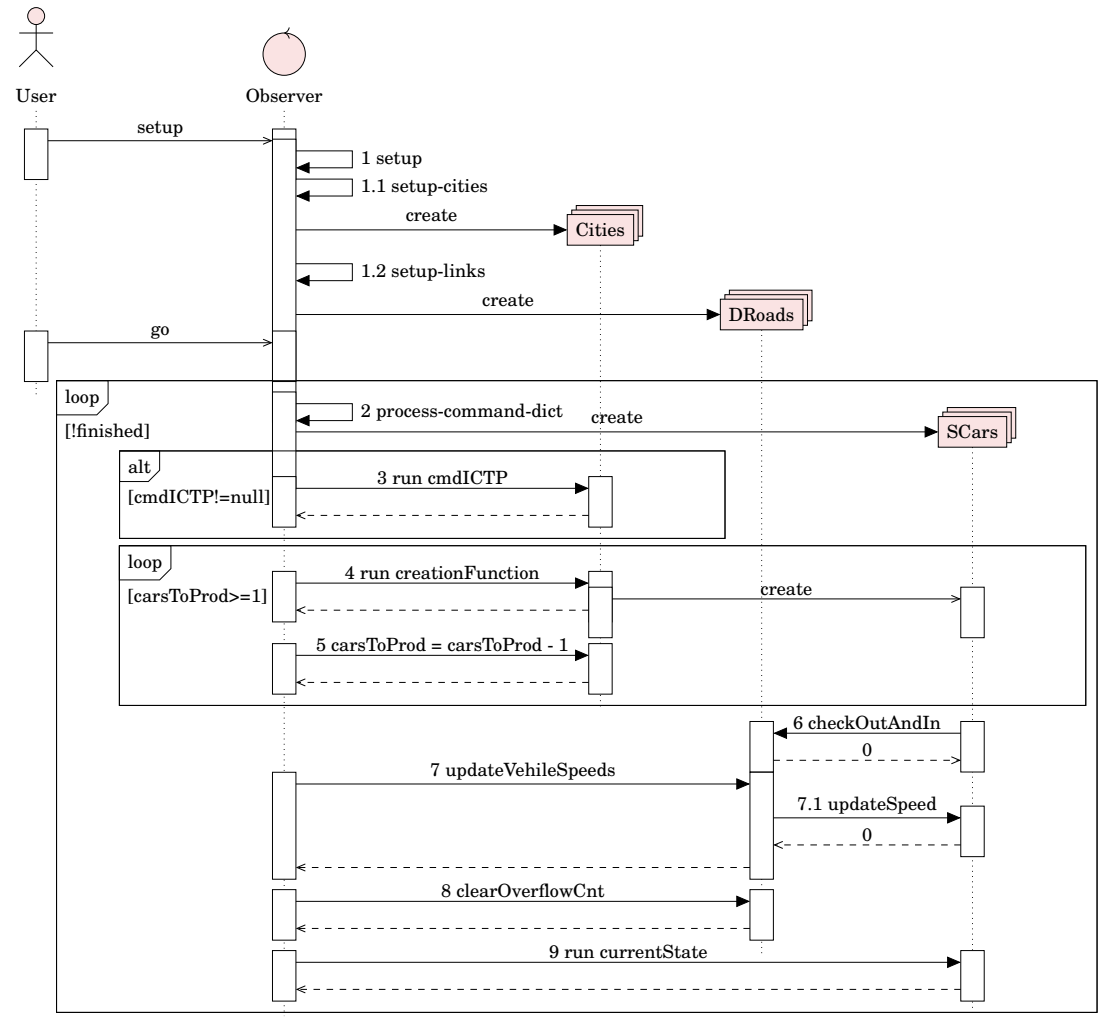


Figure 5.2: General flow of control in a NetLogo simulation created from an Athos model.

When the simulation is initialised, another user interaction is required for the simulation to commence. With the activation of the 'go button' in the simulation interface, the user triggers the simulation process. The commands in the command dictionary scheduled for direct execution are processed first. The previous section discussed how this is done via the command dictionary. As can be seen in the diagram, the execution of these commands normally results in the creation of various vehicles or *state car (SCar) agents*.

At this point, it must be noted that there are two general mechanisms that sprout state car agents into a simulation's network. The first mechanism is the command dictionary which was already discussed. The commands in the dictionary result from the definition of those `SourceNodes` that sprout agents only once

at a specified point in time. As was discussed in Section 4.3.1, Athos also allows the definition of SourceNodes that continuously sprout agents into the network. Syntactically, these nodes can be distinguished from their peers via the frequency keyword (see Figure 4.8). The definitions of these Sourcish elements for which a frequency different from zero is defined does not result in the generation of a command for the command dictionary. Instead, for these SourceNodes two anonymous commands[4] are generated: Running the first of these commands[5] (referenced by the variable commandIncreaseCarsToProduce) increases the carsToProduce counter of the SourceNode by the specified frequency value; this is done every n-th tick until a pre-defined point in time (see Figure 4.8). The carsToProduce value determines the number of cars that the source is to sprout into the simulation. In case this value is greater than one, the second anonymous command is run. This command (referenced by the creationFunction variable) creates an SCar of the defined type and decreases the value in the carsToProduce variable by one. This is repeated until the value in the carsToProduce variable is less than one.

In the generated NetLogo simulation, every DRoad has a list of all SCars currently traversing it. For this mechanism to work, it is crucial that SCars de-register themselves with the road they leave and register themselves with the one they enter, if necessary. This process always takes place when an SCar arrives at a node of its route. Every tick, DRoads recalculate their traversal speed and impose it on every vehicle in their list[6].

Another mechanism present in the NetLogo simulation is the overflow calculation. The name of this mechanism may be somewhat misleading. Its purpose is to reduce the chance of two different SCar agents occupying the same spot on a road. If more two or more agents are sprouted with the same destination, this is detected via a DRoad's overflow counter and one of the agents is nudged a few spatial units in front of the other. This counter must be reset in every tick.

[4] https://ccl.northwestern.edu/netlogo/docs/programming.html#anonymous-procedures
[5] https://ccl.northwestern.edu/netlogo/docs/dictionary.html#run
[6] In the actual code this behaviour is slightly more complex.

The final step of the simulation loop is the observer agent asking every SCar agent to run its current state. To understand this mechanism, it is important to know that agent behaviours which in Athos were modelled by means of AgentBehaviourStates with associated AgentBehaviourDescriptions (see Section 4.1.2 and Section 4.3.2) are translated into a set of NetLogo commands. This set of commands is based on the behaviour found in UML state machines. According to the UML specification, state machines exhibit an *entry* behaviour upon assumption of a new state, a *do* behaviour while being in a given state and an *exit* behaviour prior to leaving a state (OMG, 2017, pp. 320–321). To mimic this mechanism, every AgentBehaviourState of an Athos model is transformed into three commands in which the SCar's entry, do and exit behaviour is encoded. Each SCar possesses a currentState variable in which an anonymous command is stored that when run invokes one of these state machine commands. Section 5.6 will provide some further technical information on this mechanism before the transformations (code templates) are presented in the subsequent sections.

5.3 Overview on the Athos generator

Figure 5.3 illustrates a simplified and condensed call hierarchy of the template Methods used in the Athos generator. The entry point for the generator is the compile() method that requires the Model (see Section 4.1.1) root element as a parameter. Before the generator starts the actual code generation process, it needs to reset and initialise various utility class instances that are required during the generation process (this is depicted by the letter U in the illustration). These utility classes are presented in Section 5.4.

The first lines of NetLogo code produced by the generator represent static (boilerplate) infrastructure (I) code, i.e. these lines are (mostly) the same for every generated NetLogo simulation. For example, every NetLogo simulation requires a 'model head' in which the Athos optimisation extension (see Section 5.10) is loaded, the different breeds of agents and links are specified and global variables are de-

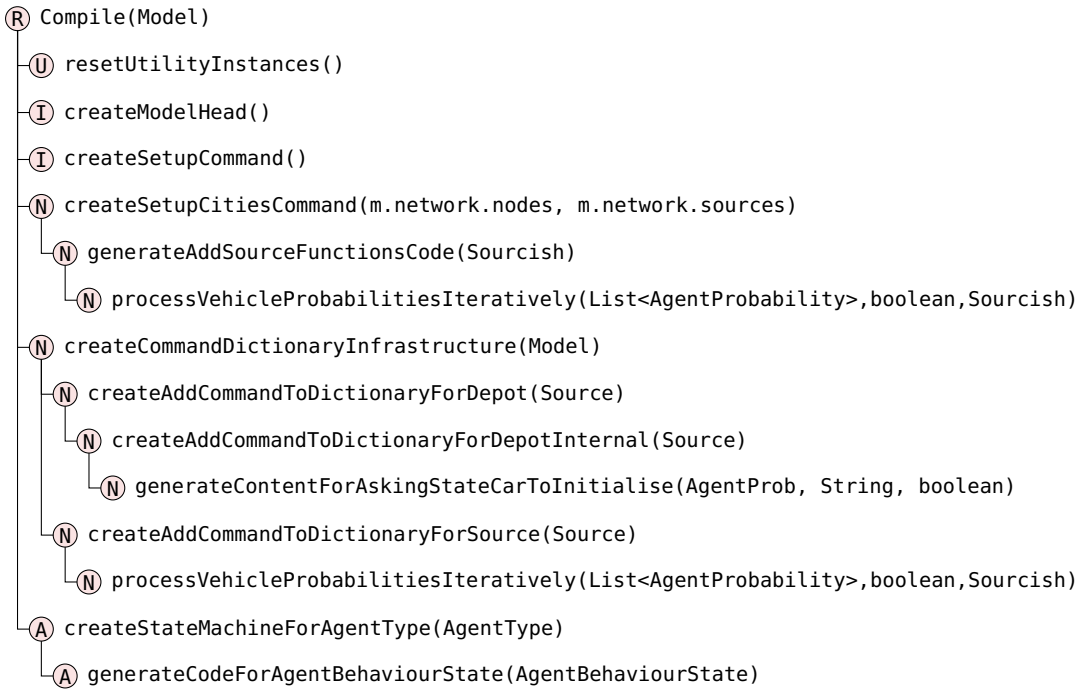


Figure 5.3: Transformations call stack of the Athos generator. The hierarchy presents the root template method (R), template methods that generate static infrastructural code (I), network-related code (N), and code for the implementation of agent behaviour (A).

clared. Another static command is the `setup` command. This command is what is executed when the setup button of the generated simulation is pressed. While the `setup` command is the same for every generated simulation, the implementation of the (sub-)commands invoked within the setup command varies as these commands use the information stored in the Athos model. Among the boiler plate code generated for every NetLogo simulation there are also NetLogo utility commands. Section 5.5 briefly discusses one such utility command to provide some deeper understanding of the generated NetLogo simulation models.

The template methods that are highly dependent on the information provided in the Athos models are those concerned with the creation of the commands that map the Athos network section to NetLogo commands which built an equivalent network infrastructure in the NetLogo model. Another set of highly dynamic transformations comprises those template methods that implement the agent's exhibited behaviour. Section 5.7 presents the details on how `AgentTypes` are used to derive appropriate

Table 5.1: Description of the most important utility classes used in the transformations together with their respective identifier.

Class	Id	Description
`CityYellowPages`	`cyp`	Provides a bidirectional mapping between the name of a node in the Athos model and its `who` number in NetLogo.
`AgentTypeYellowPages`	`aty`	Supports the creation of state-machine command names from `AgentTypes`.
`GeneratorUtil`	`gu`	Provides various generator methos especially for the genration of routing lists.
`CoordinateConverter`	`cc`	Transforms any coordinate system into one that has its origin at (0, 0) and only uses positive coordinates.

state machine commands. Section 5.8 discusses some selected template methods that are used to generate the command dictionary infrastructure with a focus on the commands created for depots of VRPTW simulations.

5.4 Utility classes used by the generator

To cope with the complexity of the transformation process, the Athos generator relies on the functionalities provided by a set of utility classes. These classes were developed to encapsulate the functionality required by different transformations. As is common in object-oriented languages, most of these functionalities are accessed via reference viariables that point to an instance of the utility class. The respective identifiers and their type are summarised in Table 5.1. For a better comprehension of the transformations presented in subsequent sections, this table might be a valuable help. It is, however, important to note that not all utility methods are accessed via this traditional approach, since the Xtend language allows for creation of *static extension methods*[7] so that in the presented transformations some meta-model elements will appear to offer generation utility functions by themselves.

Table 5.1 provides an overview of the most important utility classes used in the listings presented in this section. For every utility class, the table lists the identifier

[7] https://www.eclipse.org/xtend/documentation/202_xtend_classes_members.html#extension-methods

by which it is referred to in the listings. The table also provides a summary of how each of these classes support the code generation process.

`CityYellowPages` is a class that allows storage and retrieval of the information on how each Athos node is referenced in the generated NetLogo model. While in the Athos model every node has a unique identifier (e.g., `n1`, `n2`, etc.), in the generated NetLogo model the `who`[8] numbers of the nodes are used. It is important to note here, that in the NetLogo model, the nodes are referred to as `cities` (in other words, Athos nodes are transformed into NetLogo agents that belong to a breed[9] called cities). The `CityYellowPages` class allows a mapping in both directions, i.e. the Athos identifier can be used for retrieval the correct `who` number and vice versa.

`AgentTypeYellowPages` is a utility class that supports the transformation of `AgentTypes`, `AgentBehaviourStates` and `AgentBehaviourDescriptions` (see Section 4.1.2) into NetLogo commands[10] that allow agents to apply a state-machine mechanism for the implementation of their different behaviours. The naming patterns of these commands is implemented in a way that they are independent of the `AgentStateBehaviour` names used in the Athos model. This data structure stores the internal command names and allows their retrieval. For this, it creates a command name composed of the name of the `AgentBehaviourDescription`, the internal 'machine number' associated with the `AgentType` instance and the internal number associated with the `AgentBehaviourState` (see Section 5.7). This utility class will be discussed in more detail together with the generation of agent behaviour in Section 5.6.

`GeneratorUtil` is a utility class that supports various helpful functionalities. One example is that it can be used to determine whether a given `AgentBehaviourDescription` expects a simple list of nodes as a routing list or a list consisting of tuples. One such tuple comprises a node and a boolean value that indicates whether the agent is supposed to service or just pass through the node. It can also

[8] see `https://ccl.northwestern.edu/netlogo/docs/dictionary.html#who`

[9] see `https://ccl.northwestern.edu/netlogo/docs/dictionary.html#breed`

[10] There are two kinds of `procedures` in NetLogo: *commands* that contain a set of statements and do not return a value and *reporters* that can also contain several statements but must report a value.

Listing 5.1: Utility command used for route list initialisation.

```
;; caller: state-car
to initialise-car-with-list-auto-completion [_listOfNodes ]
  let resList []
  let finished false
  let i 0
  if current-city != item 0 _listOfNodes [
    set _listOfNodes fput current-city _listOfNodes
  ]
  while [not finished] [
    ifelse are-connected? (item i _listOfNodes) (item (i + 1) _listOfNodes)
    [
      set resList lput (item i _listOfNodes) resList
    ]
    [
      set resList sentence resList
        graphextension:fastest-path cities droads (item i _listOfNodes) (item (i + 1) _listOfNodes)
      set resList  but-last resList
    ]
    set i (i + 1)
    if  (i + 1) = (length _listOfNodes) [
      set resList lput (item i _listOfNodes) resList
      set finished true
    ]
  ]
  initialise-car-with-list resList
end
```

produce the required NetLogo code from a list of `AgentProbs` (see Section 4.3.1). The `CoordinateConverter` is a utility class applied in the transformation of coordinates. While the Athos model allows the application of any two-dimensional Cartesian coordinate system, the coordinate system in the generated simulation assumes a coordinate system with only non-negative coordinates.

5.5 NetLogo utility commands

As was mentioned at the beginning of this section, there are various static template methods defined for the Athos generator. The purpose of these boilerplate transformations is the provision of a general simulation infrastructure. This concerns information on the breeds of agents used in the simulation and their respective attributes. But there is also a need for utility-commands that are crucial during the simulation process.

Listing 5.1 shows one such important utility method that is used by several state machine commands. The presented command is used for initialisation of an agent's `routeOfNodes` list. This list contains the exact sequence of nodes an agent will visit. It is important that two consecutive nodes in this list are connected by

Listing 5.2: General command naming pattern for state machine code.

```
to <BehvrName>-M<#>-B<#>-entry
    set currentStateFinished false
    <more initialisation code>
    transStateAndRun [-> <BehvrName>M<#>-B<#>-main]
end

to <BehvrName>-M<#>-B<#>
  <Activities for behaviour>
  if <everything done>[set curStateFin true]
  if <condition> [
    transStateAndRun[
      <BehvrName-M<#>-B<#>-exit [newStateName-M<#>-B<#>-entry]
    ]
  ]
  <Activities for behaviour>
  if curStateFin [
    transStateAndRun[
      <BehvrName-M<#>-B<#>-exit[newStateName-M<#>-B<#>-entry]
    ]
  ]
end

to <BehvrName>-M<#>-B<#>-exit [nextState]
  <cleaning statements>
  <write data to tables>
  transStateAndRun nextState
end
```

an an edge or arc. Without such a connection, an agent should not be able to move from one node to the other. For modellers it would be inconvenient to be compelled to specify movement behaviours in terms of coherent routing lists. For this reason, modellers can simply specify a list of nodes that do not have to be connected by edges. For two consecutive nodes the presented method calculates the necessary intermediary nodes by means of Dijkstra's algorithm which is implemented in the Athos' optimisation library. This mechanism was also discussed in (Hoffmann *et al.*, 2018a, pp. 263–264).

5.6 Naming pattern for agent behaviour descriptions

Each `AgentType` of the Athos model is translated into a set of NetLogo commands (Hoffmann *et al.*, 2019a, p. 4) . More precisely, each `AgentBehaviourState` is translated into three commands which the agent executes via an anonymous command (lambda) stored in its `currentState` variable. This anonymous command invokes the generated state machine commands. While this mechanism is rather straightforward

in the final code, it is somewhat more intricate to define the template methods that generate it. Especially the generation of matching command definitions and command invocations is a potential source of problems.

The internally generated command names for each agent state are independent of the names given to the `AgentBehaviourState` in the Athos model. The main reason for this is the avoidance of errors resulting from incompatibilities between valid names for agent behaviours in the Athos model and the rules of valid command names in NetLogo. Therefore, the generator uses an abstract naming pattern that is shown in Listing 5.2. For every `AgentBehaviourState` in the Athos model three NetLogo commands are created representing the aforementioned entry, do, and exit behaviours for each state. Each `AgentType` has an internal number that is inserted after the capitalised 'M' in the displayed command name pattern. Every `AgentBehaviourState` defined for an `AgentType` also has an internal number that is inserted after the 'B' in the naming pattern. Together with a preceding string that represents the associated `AgentBehaviourDescription` the machine number and behaviour number define the name of the generated NetLogo command. The generated command name for the do behaviour of the `bar` agent's entry state in Listing 3.1 in Section 3.4.1, for example, is `perform-agent-await-tour-from-depot-behaviour--STATE-M1-B1`.

To store information on the correct name for each `AgentBehaviourState` associated with an agent, the services provided by the utility class `AgentTypeYellowPages` are used: this class allows storage and retrieval of the correct command names using a two-level key structure in which the first key is the `AgentType` instance and the second key the `AgentBehaviourState`. Both keys are required to store and retrieve the correct command name. For convenience, the `AgentTypeYellowPages` class can also directly produce the entire state-machine string as can be seen in Line 6, Line 21, and Line 63, of Listing 5.4.

Listing 5.3: Template methods that create the state-machine commands in NetLogo.

```
1  def createStateMachineForAgentType(AgentType agentType)'''
2    ;; ~~~~~~ MACHINE «aty.getAgentNumber(agentType)» ~~~~~~~~~~~~~~~
3    «FOR a : agentType.behaviourStates»
4    «a.generateCodeForAgentBehaviourState»
5    «ENDFOR»
6  '''
7
8  def generateCodeForAgentBehaviourState(AgentBehaviourState abs){
9    var AgentBehaviourDescription description = abs.description
10   switch description {
11     AgentAwaitTourFromDepotBehaviour:'''
12       /* TEMPLATE CODE FOR STATE-MACHINE COMMANDS
13           OMITTED FOR BREVITY */
14     '''
15     /* Agent-behaviours specified in the agent types */
16     AgentExactTourFollowingBehaviour: ''' /* OMITTED FOR BREVITY */ '''
17     AgentVanishingBehaviour: ''' /* OMITTED FOR BREVITY */ '''
18     ...
19   }
20 }
```

5.7 Agent type transformations

The `createStateMachineForAgentType()` and the `generateCodeForAgentBehaviourState()` methods are responsible for the creation of the state-machine commands used in the NetLogo simulation. Both template methods are presented in Listing 5.3: the former method (Line 1 to Line 6) processes an `AgentType` and contains a loop that iteratively calls the latter method (Line 8 to Line 20) for every `AgentBehaviourState` defined for the given `AgentType`.

In Xtend, everything inside a *template expression*, i.e. everything between the three opening apostrophes and the three closing apostrophes (’’’), will be interpreted as a string (or sequence) of characters. In Listing 5.3 this is the case for half of Line 2. These characters are displayed in blue colour. Inside a string template it is also possible to place additional code to be processed. For this, a pair of ‘guillemets’ («») is used. In Listing 5.3, Line 2 calls the `AgentTypeYellowPages` (referenced by `aty`) to provide the name for the agent type. As can be seen from Line 2 to Line 6, it is also possible to place specific control structures like loops inside such guillemets[11].

The `generateCodeForAgentBehaviourState()` template method is at the heart of the `AgentBehaviourState` to state-machine commands transformation mechan-

[11] A detailed explanation of template expressions can be found at `https://www.eclipse.org/xtend/documentation/203_xtend_expressions.html#templates`.

ism. In Line 9, the method first obtains the `AgentBehaviourDescription` associated with the state that is currently being processed. It then utilises a switch expression[12][13] to determine the *dynamic type* of the obtained behaviour description. Depending on this dynamic type, a different set of NetLogo state-machine commands (entry, do, exit) is generated.

Listing 5.4 lists a condensed excerpt of the switch-expression that transforms instances of `AgentAwaitTourFromDepotBehaviour` (a subclass of `AgentBehaviourDescription`) into NetLogo state-machine code. The lines in which the aforementioned naming pattern is relevant are highlighted in the presented listing. Note that the code displayed in blue is the NetLogo code that is generated. The code inside the guillemets («») is Xtend code, that calls other Xtend template methods which then again produce NetLogo code. The important aspect to note in the listing is that in Line 6, the name for the entry-behaviour command associated with the `AgentBehaviourDescription` is generated. Analogously, Line 21 creates the name for the NetLogo command that represents the do-behaviour, and finally Line 63 generates the name of the NetLogo command for the exit-behaviour. Line 58 to Line 63 generate the NetLogo code by which the respective agent performs its state transitions. The generated code must call the correct entry-behaviour command names for the state transitions to work.

Line 1 is the entry into the presented case of the switch-expression, i.e. the code in the listing is generated whenever the `generateCodeForAgentBehaviourState()` is passed an instance of `AgentAwaitTourFromDepotBehaviour`. The comments in Line 2 to Line 5 indicate another mechanism used in the state-machine methods: every state-car agent features a set of `descUtil` variables that the state-commands can use to store data for their state specific purposes. This means that the meaning of the data stored in these variables changes with the state the agent is currently in. For the state-commands generated for instances of `AgentAwaitTourFromDepotBehaviour` `descUtil1` and `descUtil2` are used to store

[12] This is no mistake: in Xtend everything is an expression (see Bettini, 2016, pp. 63–64).

[13] `https://www.eclipse.org/xtend/documentation/203_xtend_expressions.html#switch-expression`

Listing 5.4: Condensed excerpt of switch expression.

```
AgentAwaitTourFromDepotBehaviour: '''
  ;; For this state:
  ;; descUtil1: boolean : if true, the agent is in a blocked mode that simulates
  ;; the agent servicing a customer
  ;; descUtil2: int : counts the ticks the agent has been in servicing mode */
  to «description.nameForBehaviourDescription»-entry  ;; entry-behaviour
    set currentStateFinished false
    set descUtil1 false
    set lastCity nextCity
    set drvCnt 1
    set nextCity (item 0 (item (drvCnt mod (length routeOfNodes)) routeOfNodes))
    set currentRoadSCarAttr (droad [who] of lastCity [who] of nextCity)
    ask currentRoadSCarAttr [(run updateFunction myself 1)]
    face nextCity
    set overflow ([overflowCounter] of currentRoadSCarAttr * 0.2 *
                 ([link-length] of currentRoadSCarAttr / [linm] of currentRoadSCarAttr))
    ask currentRoadSCarAttr [set overflowCounter (overflowCounter + 1)]
    change-state-to-and-run [ -> «description.nameForBehaviourDescription»]
  end

  to «description.nameForBehaviourDescription» ;; s-car-STATE
    if is-in-city nextCity
    [
      set outputUpdateRequired true
      ifelse nextCity = (item 0 (last routeOfNodes))
        AND (item 1 (item (drvCnt mod (length routeOfNodes)) routeOfNodes))
      [
        set lastCity nextCity
        set currentStateFinished true
      ]
      [ ;; level 0 then
        if (item 1 (item (drvCnt mod (length routeOfNodes)) routeOfNodes))
          AND descUtil2 = 0
        [
          set descUtil1 (NOT descUtil1)
        ]
        if NOT descUtil1
        [
          perform-statistics-update
          increase-drvCnt
          set-next-city-of-tour-statically-lol
          update-tour-completion-statistics
        ]
      ]
    ]
    ifelse NOT descUtil1
    [
      update-my-current-speed-behaviour
      move-car
    ]
    [
          set descUtil2 (descUtil2 + 1)
          if descUtil2 >= [serviceTime] of nextCity
          [
            set descUtil2 0
          ]
    ]
    «IF abs.transition !== null && !abs.transition.isEmpty»
      «generateTransitionConditions(0,abs.transition.toList)»
    «ENDIF»
  end

  to «description.nameForBehaviourDescription»-exit [nextState]
          change-state-to-and-run nextState
  end
'''
```

Listing 5.5: Boiler plate template for state transitions.

```
1  def createCodeForTransitions()'''
2  ;;*******************************
3  ;; TRANSITIONS
4  ;;*******************************
5
6  to change-state-to[newState]
7    set currentState newState
8  end
9
10 to change-state-to-and-run[newState]
11   set currentState newState
12   run currentState
13 end
14
15 to run-current-state
16   run currentState
17 end
18 '''
```

information necessary to simulate an agent servicing a customer. In the generated simulation, the agent will halt its movements for the amount of time specified as the `serviceTime` of the customer.

Line 6 creates the state-behaviour entry command. To ensure that the command name adheres to the naming pattern discussed earlier, the `nameForBehaviourDescription()` extension method[14] provided by the `AgentTypeYellowPages` (see Table 5.1) is used. When the initial preparations are made, the entry behaviour is finished and the actual do behaviour can be executed as can be seen in Line 18. The generated call of the `change-state-to-and-run` command relies on another NetLogo utility command created by a boilerplate template method (i.e., a method that statically produces the same string whenever it is called) shown in Listing 5.5 (it is called once for every generated NetLogo simulation). Its purpose is updating the `currentState` variable of the state-car and subsequently `run` the state passed as an anonymous-command.

Line 21 marks the beginning of the do-behaviour state-machine command that is invoked when the anonymous command stored in a state-car's `currentState` variable is `run`. As was previously explained, agents are `asked` by the observer agent to `run` their current behaviour in every iteration of the simulation loop. The code in Line 58 to Line 60 maps the `AgentBehaviourTransitions` (see Section 4.1.2) of

[14] https://www.eclipse.org/xtend/documentation/202_xtend_classes_members.html#extension-methods

an Athos model to a nested `if-ifelse-else` structure in the generated NetLogo model. In order to get the syntax of this construct right, the generator must keep track on the number of the transition it currently processes and the number of total transitions so the else part is created correctly for the last transition.

5.8 Generation of the command dictionary

This section presents the template methods that generate the command dictionary infrastructure which is one of two mechanisms by which agents are created into the generated NetLogo simulations. To provide some insight on how the command dictionary works and how it is linked to the sources defined in a NetLogo network, this section will discuss the `createCommandDictionaryInfrastructure()` method and follow the call hierarchy that was elaborated on in Section 5.3.

The `createCommandDictionaryInfrastructure()` template method in Listing 5.6 creates the static NetLogo code necessary for the command dictionary to work as described in Section 5.1. Especially the boilerplate code from Line 15 to Line 61 is responsible for the implementation of this functionality.

The loops from Line 4 to Line 11 are relevant for the generation of the commands that sprout agents into the simulation. The loop from Line 8 to Line 11 is used for simple one-time agent creations like the `baz` agent in Section 3.4.2 that is once sprouted at node `n4` at tick 0 (default) and once at node `n5` at tick 1.

Agents that start as part of a fleet at a depot from which they await to be assigned a tour are sprouted from commands created in the loop from Line 4 to Line 7. The method `createAddCommandToDictionaryForDepot()` called in Line 6 generates the code that adds the necessary command for the creation of the vehicle fleet to the command dictionary.

Listing 5.7[15] shows the implementation of the `create-Add-Command-To-Dictionary-For-Depot()` template method. In Line 2, the command for the addition of

[15]The presented code was slightly simplified in comparison to the code found in the actual Athos generator.

Listing 5.6: Creation of the command dictionary infrastructure.

```
def createCommandDictionaryInfrastructure(Model m)'''
to setup-command-dictionary
  set commandDictionary table:make
  «FOR depot : m.network.sources.filter[it -> it.tourOptimisation !== null]
             + m.network.nodes.filter(SourceNode).filter[tourOptimisation !== null]»
    «depot.createAddCommandToDictionaryForDepot»
  «ENDFOR»
  «FOR source : m.network.sources.filter[it | it.simpleStart]
              + m.network.nodes.filter(SourceNode).filter[simpleStart]»
  «source.createAddCommandToDictionaryForSource»
  «ENDFOR»
  recalculate-command-processing-infrastructure
end

to add-command-to-command-dictionary-without-recalculation[executionTime command]
  ifelse table:has-key? commandDictionary executionTime
  [
    let listOfCommands table:get commandDictionary executionTime
    set listOfCommands lput command listOfCommands
    table:put commandDictionary executionTime listOfCommands
  ]
  [
    table:put commandDictionary executionTime (list command)
  ]
end

to add-command-to-command-dictionary[executionTime command]
  add-command-to-command-dictionary-without-recalculation executionTime command
  recalculate-command-processing-infrastructure
end

to recalculate-command-processing-infrastructure
  set currentExecutionTimeList table:keys commandDictionary
  ifelse not empty? currentExecutionTimeList
  [
    set currentExecutionTimeList sort-by < currentExecutionTimeList
    update-currentExecutionTime-currentExecutionTmeLst-currentCmdLst-commandDictionary
  ]
  [
    set currentExecutionTimeList (list )
    set nextExecutionTime -1
  ]
end

to update-currentExecutionTime-currentExecutionTmeLst-currentCmdLst-commandDictionary
  if not empty? currentExecutionTimeList
  [
    set nextExecutionTime first currentExecutionTimeList
    set currentExecutionTimeList remove-item 0 currentExecutionTimeList
    set currentCommandList table:get commandDictionary nextExecutionTime
    table:remove commandDictionary nextExecutionTime
  ]
end

to process-command-dictionary
  if nextExecutionTime >= 0 AND ticks = nextExecutionTime
  [
    foreach currentCommandList [x -> run x]
    update-currentExecutionTime-currentExecutionTmeLst-currentCmdLst-commandDictionary
  ]
end
'''
```

Listing 5.7: Addition of a creation command for a depot.

```
1  def createAddCommandToDictionaryForDepot(Sourcish source)'''
2    add-command-to-command-dictionary-without-recalculation «source.at» [
3      [] -> ask city  «cyp.getCityNumber(source.sourcishName)»
4        [
5          let carLengthRelativeToStreet 0
6          if newCarPipeline = 0 [set newCarPipeline (turtle-set )]
7
8          let customersOfRoute_ownedByCity
9            (list
10             «FOR node : (source.tourOptimisation.route)»
11               city «cyp.getCityNumber(node.name)»
12             «ENDFOR»
13           )
14
15         let listOfTours graphextension:pareto-capacitated-lol-vrptw
16             cities droads (city «cyp.getCityNumber(source.sourcishName)»)
17             customersOfRoute_ownedByCity
18             «source.sproutFunction.agentProbabilities
19                 .head.agentTypeForAgentProb.maxWeight»
20             «if(b.popsize!=0){b.popsize}else{'''20'''}»
21             «if(b.simplePermuProb!=0){b.simplePermuProb}else{'''0.9'''}»
22             «if(b.maxDistance!=0){b.maxDistance}else{'''4'''}»
23             «if(b.generations != 0){b.generations}else{'''80'''}»
24             «if(b.weightNoOfTours!=0){b.weightNoOfTours}else{'''100'''}»
25             «if(b.weightTotalDistance!=0){b.weightTotalDistance}else{'''0.001'''}»
26             «if(b.tournamentSize!=0){b.tournamentSize}else{'''4'''}»
27             «if(b.takeBestProb!=0){b.takeBestProb}else{'''0.8'''}»
28             «if(b.mutationProb!=0){b.mutationProb}else{'''0.1'''}»
29             (city «cyp.getCityNumber(source.sourcishName)»)
30
31         foreach listOfTours [
32           tour -> ask patch-here
33           [
34             sprout-state-cars 1
35             [
36               «generateContentForAskingStateCarToInitialise(
37                   source.sproutFunction.agentProbabilities.head,"tour",true)»
38
39             ]
40           ]
41         ] ;; end foreach
42       ]
43   ]
44 '''
```

anonymous commands to the dictionary is invoked and passed two parameters: the time at which the command is to be executed and the command itself which spans from Line 2 to Line 43.

The generated command `asks` the depot at which the fleet of vehicles is located to execute the code in the brackets that open in Line 4 and are closed in Line 42. From Line 8 to Line 13 the depot saves its list of customers in a local variable. This list is then used in a call to the *Athos optimisation library* accessed via NetLogo's extension mechanism. The algorithm that is invoked in Line 15 is presented in great detail in the next section. The algorithm from Athos' optimisation library is an evolutionary algorithm that needs to be passed several parameters related to the

Listing 5.8: Template for the initialisation of an agent.

```
1  def generateContentForAskingStateCarToInitialise(AgentProb ap,
2      String routeString, boolean listOfLists)'''
3    «var AgentType at = ap.agentTypeForAgentProb»
4    set agentType "«at.name»"
5    set congestionFactor «at.congestionFactor»
6    set capacity «at.maxWeight»
7    set cityOfOrigin current-city
8    perform-mandatory-computations
9    «IF at.attributeAssigments !== null»
10   «FOR attrib : at.attributeAssigments»!
11   set «attrib.attribute.name» «attrib.value»
12   «ENDFOR»!
13   «ENDIF»
14   «IF at.individualOptimization»set individualFunction [ !
15         [] -> ask droads [ «at.function.name» myself]]
16   «ENDIF»!
17   «IF listOfLists»
18   initialise-car-with-list-lol «routeString»
19   «ELSE»
20   initialise-car-with-list-auto-completion «routeString»
21   «ENDIF»
22   change-state-to-and-run[ ->
23     «aty.getEntryNameForBehaviourDescription(
24       at.behaviour.agentBehaviourStates.head.description)»]
25 '''
```

problem to be solved such as the set of all cities in the network and the connecting edges, the list of customers, or the maximum capacity of the vehicle. It also requires several parameters that affect the execution of the algorithm such as the number of iterations (number of generations) or the probability for mutations to occur.

The algorithm returns a list of tours, i.e. a list in which lists of tuples are stored. Each of these tuples consists of a node that the agent must navigate to, together with a boolean value that indicates whether this node is a customer to be serviced or just a pass-through node. The list of tours is stored in the local variable `listOfTours` in Line 15. It is then used in the code from Line 31 to Line 41. For every tour in the list of tours an agent is created that will service the respective tour. This is done in the `generateContentForAskingStateCarToInitialise()` method in Line 36 and Line 37.

The `generateContentForAskingStateCarToInitialise()` is presented in Listing 5.8. Here, Line 3 to Line 16 represent code that is used to initialise various attributes of the created agent (such as the congestionFactor). Agents can also have individual attributes and optimisation functions that they intend to optimise, for which the code from Line 10 to Line 12 and Line 14 to Line 16 is necessary. Of special

importance are Line 22 to Line 24: with the invocation of the `change-state-to--and-run` command, the agent is programmed to store an anonymous command in its `currentState` variable that when `run` will invoke the entry state-machine command which is associated to the first `AgentBehaviourState` (entry state) defined for the agent in the agent type section of the Athos program. In the given example, this is the state-machine command generated for the `AgentAwaitTour` instance discussed in Section 5.7. It is implemented by the code given in Listing 5.4.

5.9 Summary

This section has shown how Athos models are transformed into executable NetLogo simulations. Section 5.1 introduced the command dictionary that is used to store and execute agent creation commands. Section 5.2 then gave insight into the general flow of control that governs the execution of any NetLogo model generated from Athos. After that Section 5.3 gave an overview on a selected set of template methods in the Athos generator that transform Athos models into NetLogo code. To facilitate comprehension of these template methods, the most important utility classes used in the template methods were introduced in Section 5.4. An example for a utility command that is used in the NetLogo code was given in Section 5.5. Section 5.6 then elaborated on the naming pattern for state-machine commands that are used in the generated NetLogo simulations to implement agents' behaviour. A concrete example how these state-machine commands are derived from Athos models was then discussed in Section 5.7. Section 5.8 concluded this section by showing how optimal tours are determined by calling Athos' optimisation library via a NetLogo extension and how the provided solution is further processed in the NetLogo code.

5.10 The optimisation library

This section presents Athos' optimisation library that features algorithms related to the optimisation of vehicle routing problems (VRPs). In order to provide some

comprehensible insight into the problems the algorithms in this library deal with, a solution algorithm for vehicle routing problems with time windows (VRPTWs) will be discussed. The algorithm implemented for Athos is based on the work of Ombuki *et al.* (2006) and was also presented in (Hoffmann *et al.*, 2019a) and (Hoffmann *et al.*, 2019b). The implementation is based on the work of Ombuki *et al.* (2006) because Ombuki et al. provide a clear and comprehensible explanation that enabled the author of this thesis to implement the algorithm in Java. Compared to the usage of pre-implemented algorithms, this allows for additional flexibility required in later versions of Athos that might allow definitions of further restrictions for the desired solutions (see 8.3.4). However, the Athos architecture also allows to use pre-defined third-party algorithms.

Although this section focuses on the evolutionary algorithm for the solution of VRPTWs, it is not the only algorithm created for the optimisation library. In (Hoffmann *et al.*, 2018a) an implementation of ant-colony system (ACS) (Dorigo and Gambardella, 1997) is discussed. In (Hoffmann *et al.*, 2020), an extension mechanism found in the Athos library was presented. This mechanism can be used to model additional, not natively supported problems with Athos. It is however not presented here, as it renders the resulting programs comparatively difficult to create, read and understand. The definition of the language elements enabling this mechanism are part of the concrete syntax presented in Appendix A.

5.10.1 General structure and access

In Section 3.3, it was explained that Athos is a language intended to be platform-independent and thus transformable to arbitrary solution domains. For this to work, it is important that the optimisation algorithms be decoupled from any target platform. The approach taken by Athos to achieve this decoupling is shown in Figure 5.4 for the evolutionary algorithm that solves instances of the VRPTW. The implementation uses three template parameters `V`, `E` and `C`. `V` represents the nodes of the network. In NetLogo, these nodes are specialisations of `Turtles`. `E` represents

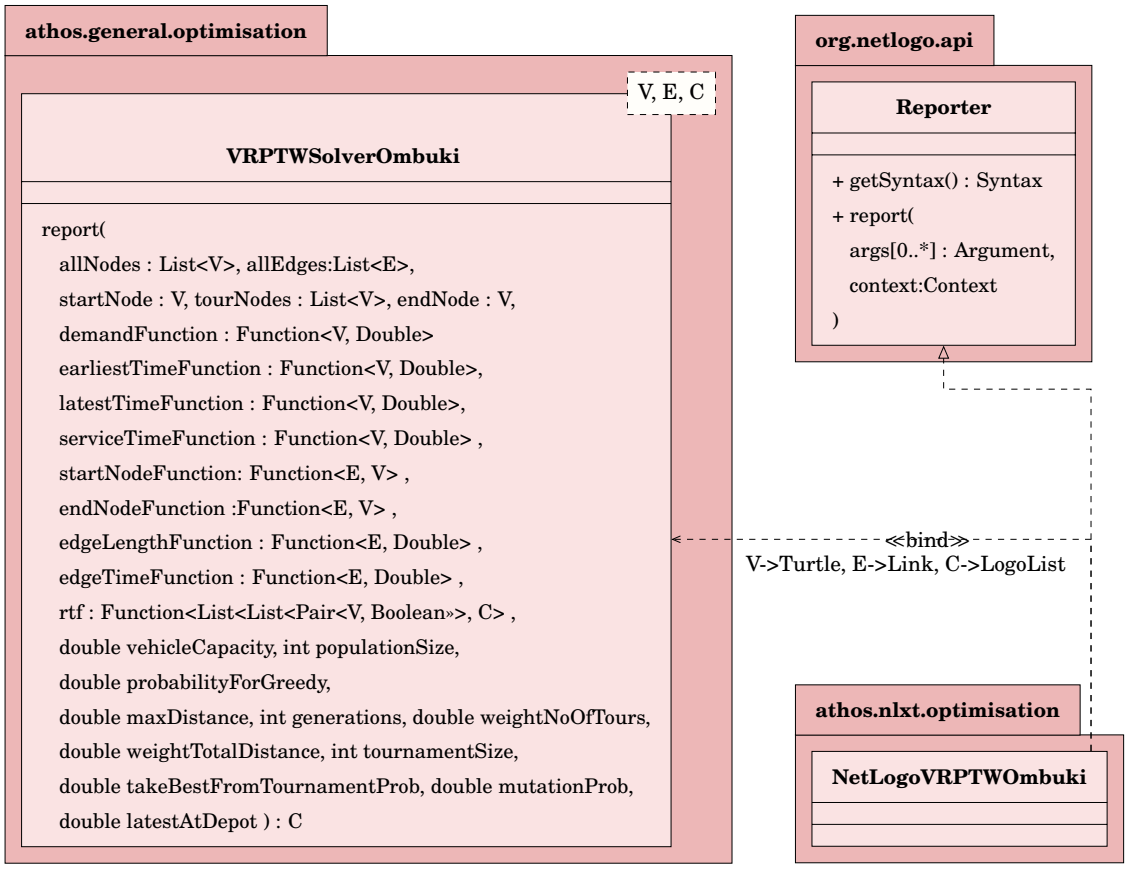


Figure 5.4: Platform independent access to the optimisation library.

the edges between nodes. In NetLogo the edges are implemented as `Links`. Finally, `C` represents the list data structure that stores the solution of the algorithm and thus is the type of object the algorithm returns. For NetLogo this is a `LogoList`.

In order to leverage NetLogo's extension mechanism[16], an implementation of NetLogos `Reporter` interface was created. This way, the implementing class `NetLogoVRPTWOmbuki` is passed the `Turtle` and Link instances of the simulation together with an array of additional parameters. The generated NetLogo command that invokes the reporter implementation was shown in Listing 5.7 of Section 5.8.

`NetLogoVRPTWOmbuki` then calls the platform-independent `report` method of `VRPTWSolverOmbuki` and binds the template parameters to its `Link`,`Turtle` and `LogoList` classes. In order for the algorithm in `VRPTWSolverOmbuki` to make sense of these classes, `NetLogoVRPTWOmbuki` has to provide several additional `Functions`, the algorithm can use to extract the necessary data from NetLogos data types. This approach provides the necessary flexibility for further development of the algorithms

[16] `https://ccl.northwestern.edu/netlogo/docs/extensions.html`

in future versions of Athos. It also grants access to additional agent properties by a simple provision of an additional function. The actual implementation code is given in Appendix B both for the implementation of the algorithm in Section B.1 and also for the invocation in Section B.2.

5.10.2 Genetic optimisation algorithm

This section discusses how the evolutionary algorithm processes the data provided via the extension mechanism in order to find an optimal solution for a VRPTW.The general process by which the algorithm tries to find an optimal solution can be outlined as follows:

1. Built the data structures (network, adjacency matrices).
2. Create initial sequence of nodes (partly randomly, partly heuristically).
3. Derive actual tours from the initial node sequence.
4. While not number of iterations performed.
    a) Stratify (calculate Pareto ranks of) the population.
    b) Find best individual in the population and add it to the next generation.
    c) Built the mating population.
    d) Add child from parents in the mating population to the next generation.
5. Return the best individual.

The next sections will go into detail on how exactly the outlined steps are performed. The explanation is done with two complementary approaches. Section 5.10.2.1 provides a walk-through for a process-centric pseudo-code formulation that is based on the one presented in (Hoffmann *et al.*, 2019a). Section 5.10.2.2 puts a stronger focus on the data flow during the execution of the algorithm.

The algorithm starts out with a set of randomly generated sequences in which the customers are ordered. It then partitions each sequence into a set of feasible tours (if this is possible) and then performs a pre-defined number of iterations in which successively new generations of tours are built. The best individual of each generation is guaranteed to be present in the next generation. Otherwise, the new generation is built from combining two parents from the previous generation. Finally, the best individual from the most recent generation is returned.

5.10.2.1 Process-centric explanation explanation

Lines 1 – 3 The algorithm processes a graph $G(V,E)$ composed of a node set V and a set of edges E. V comprises both the set of all customers $C \subset V$ and the depot d, i.e. $V = C \cup \{d\}$. Function d_f assigns a *spatial* distance to each edge in E. Function t_f assigns a *temporal* distance to each edge in E (representing the time that the vehicle requires to traverse the respective edge). Every customer node has to be *visited* exactly once by an arbitrary agent of a homogeneous set of agents who all have a maximum capacity of c_{max}. All agents start their tour at the depot d. Customer nodes possess a demand assigned to them by the function h_f. In addition, functions e_f and l_f are used to assign an earliest time (e_f) and a latest time (l_f) to the customers. This way every customer is assigned a time window in which the visit of the agent must commence. Note that l_f also assigns a latest time to the depot that represents the point in time until which all vehicles must have returned to the depot. Finally, s_f is a function that assigns a service time to every customer in C.

Lines 5 – 6: The distance and time functions d_f and t_f are used together with Dijkstra's algorithm to calculate the shortest spatial and temporal distances between any two customer nodes of the graph. These distances are stored in a `timeMatrix` and a `distanceMatrix`, respectively. Note that in complete graphs the temporal and spatial distance between any two customer nodes are equal to the values that d_f and t_f yield for the edge connecting the respective customers. In incomplete graphs, Dijkstra's algorithm must be applied to determine the shortest and fastest route between nodes that are not directly connected.

Lines 7 – 26: Having created the time and distance matrices, the initial population of chromosomes is to be created. The size of the population is determined by the parameter `popSize`. Two different strategies for the creation of chromosomes are implemented. The first strategy simply creates a random permutation of all customer nodes to visit (start and end node are always fixed and determined via parameters).

Listing 5.9: Pseudocode of the evolutionary algorithm.

```
1  Input: G(V,E), d_f : E → ℝ, t_f : E → ℝ, C ⊂ V, d ∈ V \ C, c_max ∈ ℝ
2         h_f : C → ℝ, e_f : C → ℝ, s_f : C → ℝ, l_f : C ∪ d → ℝ
3  Output: List of routes R_Tours
4  begin
5  Derive distance matrix D_ij, i,j ∈ C ∪ d using Dijkstra's and d_f
6  Derive time matrix T_ij, i,j ∈ C ∪ d using Dijkstra's and t_f.
7  let P := ∅; // Create initial population P
8  while |P| < popSize
9    if rand(0,1) < simplePermuProb
10     create a random permutation p of elements in C and add p to P;
11   else
12     set C_copy ← C;
13     init p as empty chromosome;
14     set i ← 0
15     while C_copy ≠ ∅ do
16       set p[i] ← getAndRemoveRandomElement(C_copy);
17       while ∃c ∈ C_copy : D_p[i],c < maxDistance do
18         set c_nearest ← take customer from C_copy with min{D_p[i],customer};
19         set i ← i + 1
20         set p[i] ← c_nearest ;
21       od
22       set i ← i + 1
23     od
24     add p to P
25   fi
26 od
27 set Ω ← ∅
28 foreach p ∈ P do // transform every chromosome into a list of tours!
29   set R_tours ← ∅ // the list of tours
30   add R_tours to Ω
31   set r[] ← newEmptyTour // next tour to derive
32   add r[] to R_tours
33   for i ← 0 to |p| − 1 do
34     if customer p[i] can be visited without violation of any constraint
35       add p[i] to r[]
36     else if costomer p[i] can't be reached and r[] is empty
37       ⇒ infeasible problem
38     else
39       set r[] ← newEmptyTour // start a new tour
40       add r[] to R_tours
41       set i ← i − 1
42       continue // check whether customer can be visited as first customer
43     fi
44   od
45 od
46 foreach R_tours ∈ Ω do
47   Try to improve total distance by adding last customer of route r_i[] to route r_i+1[].
48 od
49 for i ← 0 to generations − 1 do
50   foreach R_tours ∈ Ω do
51     calculate pareto rank for R_tours;
52     set w_Σ(R_tours) ← w_1 · |R_tours| + w_2 · totalDistance(R_tours)
53   od
54   set M_Mating ← ∅; T_Tournament ← ∅; Ω_NextGen
55   add R_tours from Ω to Ω_NextGen with min{rank(R_tours) then w_Σ(R_tours)} // champion
56   while |M_Mating| < 2·(popsize−1) do
57     set T_Tournament ← randomly select tournamentSize R_tours from Ω
58     if rand(0,1) < takeBestProb
59       add R_tours from T_Tournament to M_Mating with min{rank(R_tours) then w_Σ(R_tours)}
60     else
61       add all T_Tournament to M_Mating
62     fi
63   od
64   while |M_Mating| > 0
65     offspring ← take 2 R_tours from |M_Mating| and perform pairwise BCRC crossing
66     if rand(0,1) < mutationProb do mutate(offspring) od
67     add offspring to Ω_NextGen
68   od
69   set Ω ← Ω_NextGen
70 od
71 return R_tours from Ω to M_Mating with min{rank(R_tours) then w_Σ(R_tours)}
72 end
```

The second strategy selects a random customer as the first gene in the chromosome. If there is at least one customer that has not yet been visited (i.e. is not yet part of the chromosome to be created) and whose distance to the last customer (i.e. the one most recently added to the chromosome) is below the `maxDistance` threshold, the customer with the minimal distance is added to the chromosome. If none of the remaining customers' distance is below the `maxDistance` threshold, a random customer is added to the chromosome and the greedy search for the next customer to be added starts anew. This process is repeated until there is no customer left to be added to the chromosome.

Lines 27 – 48 Each chromosome p of the initial population P is now transformed into a list of tours $\mathscr{R}_{tours}$ in which every tour $r_i[]$ represents a list of customers. The set of all lists of tours is denoted as Ω. To transform a chromosome into a list of tours, the customers in this chromosome are processed in sequential order. At the beginning, an empty tour is created. For each customer it is checked, whether it can be visited within the defined constraints given the current state of the current tour. If a visit within the defined constraints is possible, the customer is added to the current tour. If a visit cannot be performed without violating one or several constraints, the current tour is closed and a new empty tour is created. It is then checked whether the customer can be visited as the first customer of the tour. If no customer can be visited at the very beginning of a tour, the problem is infeasible: if a vehicle cannot arrive in time when the customer is the first on its tour, it is simply not possible for any vehicle at the depot to visit the customer within the defined constraints. The same holds true for the demand that any vehicle can meet. After all chromosome have been transformed into a list of tours, a second phase starts. This phase attempts to improve on the accumulated distance covered by all vehicles of the list of tours. To this end, the last customer of tour $r_i[]$ is placed as the first customer of the subsequent tour r_{i+1}. If this results in a reduction of distance without constraint violation, the customer movement is made permanent. Otherwise the movement of the customer is undone.

Lines 49 – 70 Each iteration of the main loop of the algorithm generates a new generation of lists of tours $\mathscr{R}_{tours}$. The exact number of generations is defined by the `generations` parameter. In Line 51) the current generation of lists of tours is stratified by calculation of a Pareto rank for every list of tour $\mathscr{R}_{tours}$ in the current generation Ω. The stratification process is done by filtering for all non-dominated lists of tours in the current generation. A list of tours is non-dominated if there is no other list of tours in the current generation that contains a lower (or equal) number of tours with a lower (or equal) total distance (with either the number of tours or the distance being actually lower). (see (Ombuki *et al.*, 2006, p. 22)). Every list of tours in this filtered set is assigned rank 1 and the set is then removed from the ranking space. This process is repeated with ascending rank numbers (lists of the next filtered set are assigned rank 2 and so on) until the ranking space is empty. Having assigned a rank to each list of tours, Line 52 calculates a weighted sum w_Σ for each list of tours. Parameter w_1 determines the weight of the number of tours and parameter w_2 determines the weight of the total distance of the tours. Hence, the weighted sum w_Σ is defined as the sum of w_1 multiplied by the number of tours ($w_1 \cdot$ `cntTours`$(\mathscr{R}_{tours})$) and w_2 multiplied by the total distance of all tours ($w_2 \cdot$ `totalDistance`$(\mathscr{R}_{tours})$.

Lines 54 – 63: This section prepares the creation of the next generation of list of tours $\Omega_{NextGen}$. The first step is the selection of the champion (determined by rank and then by weighted sum value w_Σ) of the current generation. This champion is directly added to the next generation (Line 55). Next, a set of mating lists of tours is initialised. To populate this set, a tournament selection mechanism is applied. This mechanism randomly selects `tournamentSize` (e.g. 4) elements to form the tournament set $\mathscr{T}_{Tournament}$. A random number then determines whether all members of $\mathscr{T}_{Tournament}$ are added to the mating population M_{Mating} or only the best element in $\mathscr{T}_{Tournament}$ (again determined by rank and then by weighted sum value w_Σ) is selected for reproduction. The outcome of this randomised decision can be influenced via the `takeBestProb` parameter (Line 58). This tournament selection

and randomised addition is repeated until the number of elements in the mating population exceeds `popsize` – 2 elements (lists of tours).

Lines 65 – 71: Finally, two lists of tours $\mathscr{R}_{Tours}$ at a time are taken from the mating set M_{Mating} and crossed by means of the best cost route crossover (BCRC) operator (Ombuki *et al.*, 2006) (also see Appendix B). This crossing of two parent lists of tours results in an offspring list of tours o. Based on the outcome of another random experiment (influenced by the parameter `mutationProb`), a mutation process (see Ombuki *et al.*, 2006) might be imposed on offspring o. The current implementation of this evolutionary algorithm yields the champion of the last generation (determined by w_Σ as the optimal solution.

5.10.2.2 Data-centric explanation

Figure 5.5 presents the algorithm with a stronger focus on the data flow. The figure shows how the first action is the creation of a bidirectional mapping between the customer object and an integer that represents the customer in the algorithm. This bidirectional map is represented as a data store. In order to pass its data to actions, its linked to connector `A`. Another bidirectional map stores data on the customer such as the time window and the service time (`B`). The nodes and edges that are passed to the algorithm as parameters (depicted on the left hand side on the edge of the box) are used to create a graph structure (`C`). Together with the bidirectional id map, this graph is used to built a distance map (`D`) and a time map (`E`). After that, the creation of the chromosome population is performed. For this step, the population size (`popSize`) and two parameters that affect *if* (`simpleProb`) and *how* (`maxDist`) the chromosomes are created with a greedy approach, or a random ordering. The created chromosome population is kept in another data store (`F`).

The start node and end node parameter (those were not part of the chromosome creation process as their position as the first and last element are fixed), the data on the customer, and the distance and time matrix are used to turn the chromosome

population into a list of tours[17]. For each list of tours the accumulated length of all its tours is stored in the ToursLengthMap (H). The lists of tours are then stratified by calculation of Pareto ranks based on the total tour length and the number of vehicles applied. The stratification result can be accessed via the data store associated with connector (I).

In the next action, a weighted value is calculated based on the weight (wghtDist) that was ascribed to the number of tours (P4) and the weight ascribed to the covered distance (wghtDist). For each tour its weighted sum is also saved in a data store (J). In the next step, the mating population is prepared and filled. For this, the rank (value obtained in the stratification process) is used together with the weighted sum of the tour.

A tournament group of a size determined by the tournament size parameter trnmtSize is created by random selection of tournament participants. Then, it is randomly decided whether all tournament participants or only the best enter the mating population. The threshold for this randomised decision can be controlled with the takeBest variable. In casewhere the best list of tours is to be selected, the data from the stratification and weighting process (J) are used. The mating population (L) is then used to breed a new generation. Some of the new offspring list of tours will be subjected to a mutation process. The chance by which this mutation process is carried out depends on the value of the mutProb parameter. This process is repeated until the number of generations specified via the noGens parameter are created. The best list of tours (based on the ranks and weighted sum value) is the champ. The list of tours is modified in a way so that it is turned in to a sequence of nodes in which two subsequent nodes are always connected by an edge. The modified champion is the result of the algorithm.

[17] Due to the limited space, the illustration only uses the term 'tours' instead of the more appropriate 'list of tours'.

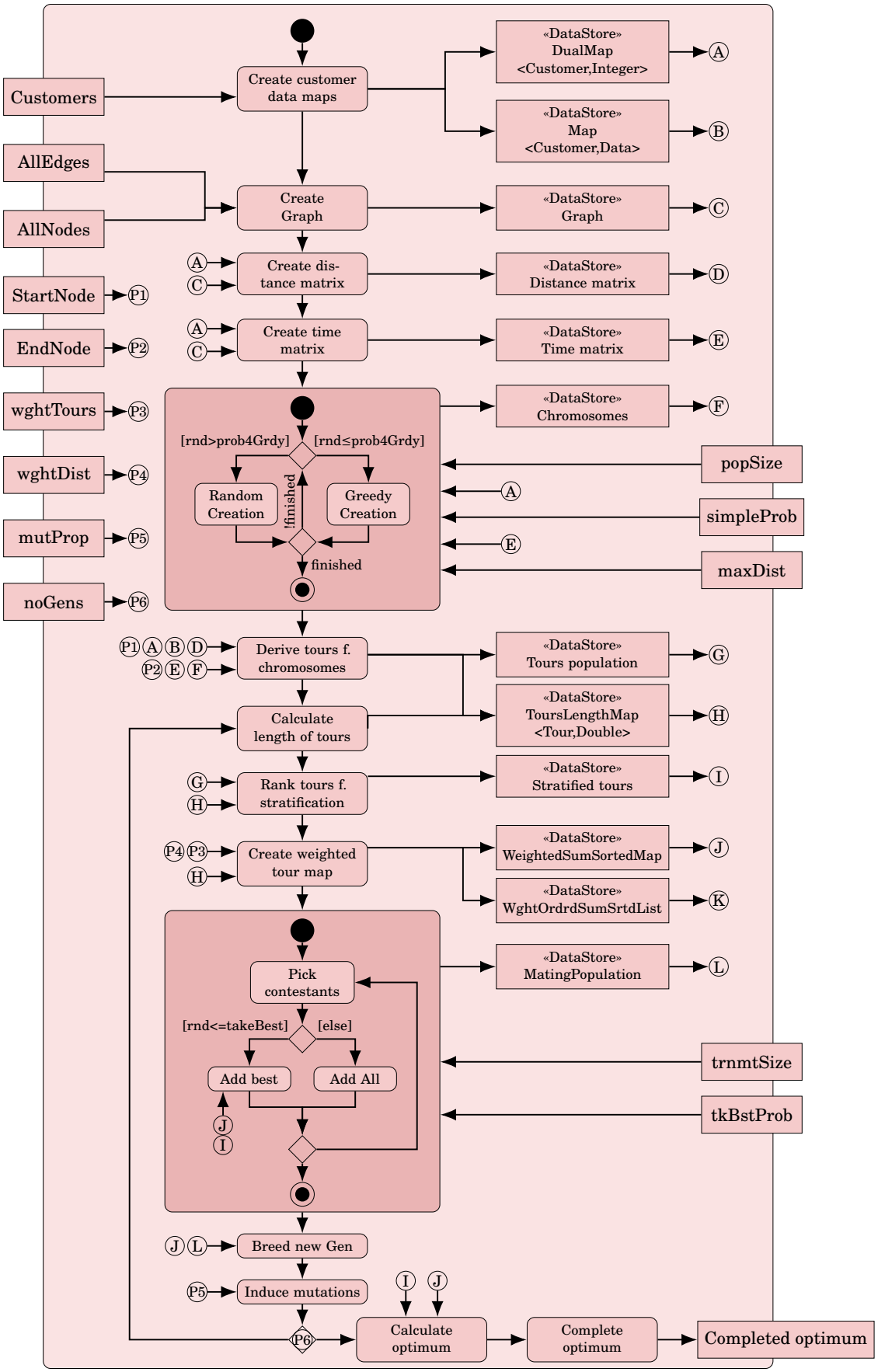


Figure 5.5: The control flow of the evolutionary algorithm implemented in the Athos optimisation library. The control and data flow is based on the work of (Ombuki *et al.*, 2006).

Appropriate values for the presented control parameters must be discovered by empirical observations. Ombuki *et al.* (2006) state the following value for the control constants:

- popSize: 300
- simplePermuProb: 0.9
- maxDistance: to be empirically decided
- generations: 350
- takeBestProb: 0.8
- tournamentSize: 4
- mutationProb 0.1

It is to be noted that the `WghtOrdrdSumSrtdList` (`K`) is currently not used. Its purpose was the provision of insight and support for further analysis of the solution space. For an accelerated runtime-behaviour of the algorithm, the creation and updating of this data structure can be removed.

5.10.3 Performance evaluation

This section briefly discusses the performance of the algorithm in terms of the quality of the obtained solutions and the number of iterations required to obtain the respective solution. In order to evaluate the presented algorithm, it is compared to another evolutionary algorithm as a baseline approach. A well-known and widely applied algorithm for the solution of VRPTWs can be found in the JSprit optimisation library[18]. The two algorithms are compared by an analysis of the solutions they produce on a selection of benchmark problems. The analysis will focus on a comparison of the best solution produced by the algorithms in each iteration.

For the comparison of the algorithms, the Solomon's benchmark problems for VRPTWs will be used. Both algorithms will assume an objective function that weighs the number of deployed vehicles with a factor of 10 whereas the travelled distance is weighted with a factor of 0.1. The decision for these values seems sensible as each additional vehicle certainly results in considerably higher costs than an

[18] `https://jsprit.github.io/index.html`

additional unit of distance to be travelled. However, the exact values of 10 and 0.1 were arbitrary and are not based on any scientific standard.

Other than the weight values for the objective function and the setting of the 'FAST_REGRET' parameter to true (which is supposed to reduce computation time) no additional parameterisation was done for the JSprit algorithm. The values for the control variables of the Athos algorithm are those proposed by Ombuki *et al.* (2006) given in the previous section. The *maxDistance* control variable was set to 90. Both algorithms performed exactly 125 iterations on the Solomon C101, C201 and the RC201 problems that feature exactly 100 customers each. The former two problems feature clusters of customers, i.e. areas in which many customers are close to each other. The latter problem features both clusters of customers and customers randomly dispersed over the respective map[19]. After each iteration, the objective value produced by the best known individual in the respective solution space was recorded. For each problem instance this was done 25 times.

Figure 5.6 presents the results of the performed experiments. Each of the six plots features three graphs: one graph representing the minimal objective value found among the 25 executions in the respective iteration, one for the analogous maximum value and one for the average. The dashed blue line drawn in each of the six plots shows the objective value of the best known solution for the respective problem[20].

What can be clearly seen from the graph is that both the Athos and the JSprit algorithm converge towards the optimal solution so that after 125 iterations both algorithms provide solutions that are either optimal or near optimal. However, it must also be noted that the JSprit algorithm produces very stable results, i.e. the results for each of the 125 iterations were similar among the 25 executions. Moreover, it can also be seen that the JSprit algorithm requires less iterations to achieve a near optimal or optimal solution. Two additional problems with the algorithm

[19]This is also explained on the website of the creator of the problems: `http://web.cba.neu.edu/~msolomon/problems.htm`

[20]This value was taken from `https://www.sintef.no/projectweb/top/vrptw/100-customers/`. Note that the objective value is to be calculated by multiplying the number of vehicles with a factor of 10 to which the distance divided by ten is to be added.

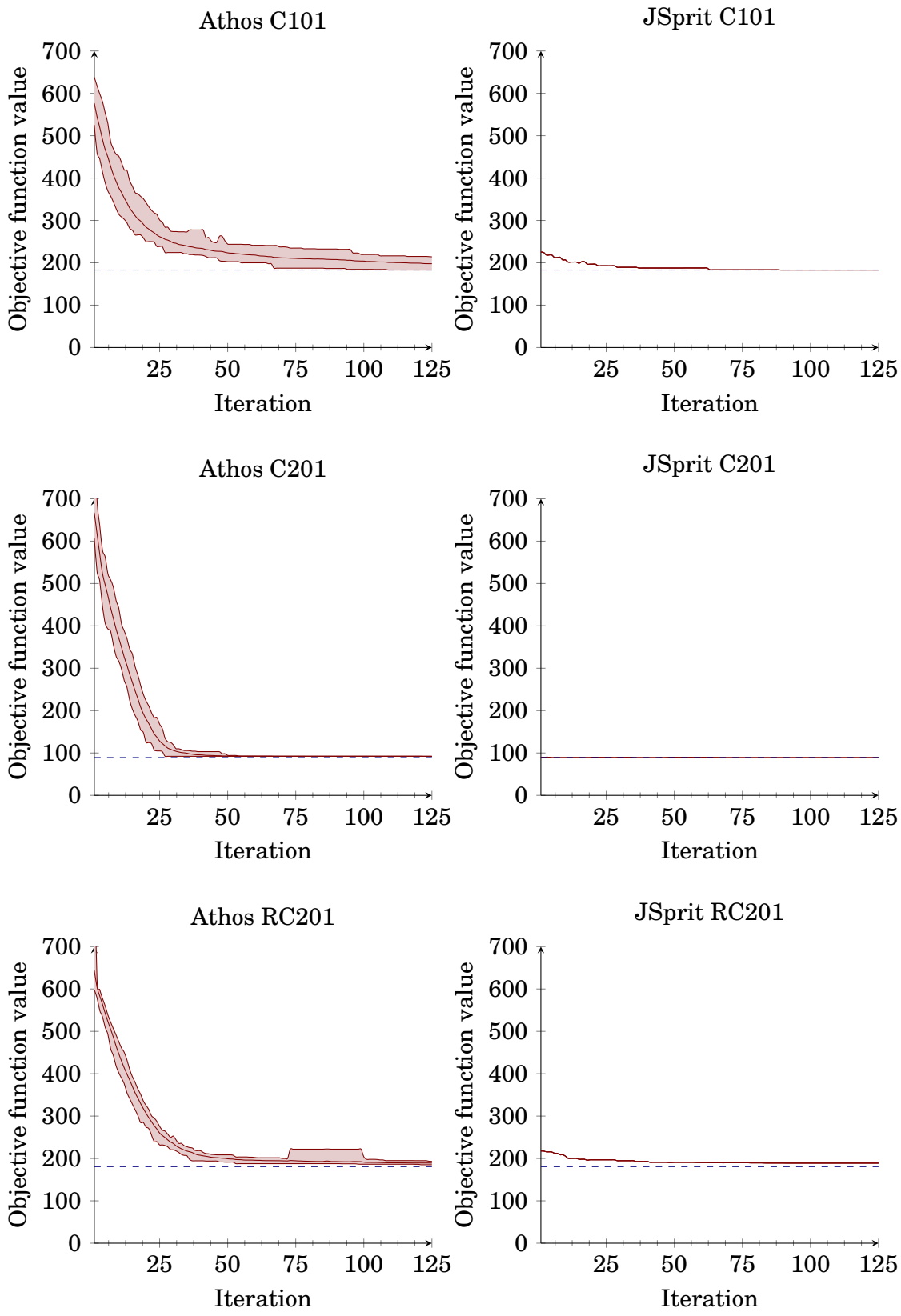


Figure 5.6: Per-iteration comparison of the best solutions produced for a selection of Solomon's VRPTW benchmark problems by Athos' and JSprit's respective evolutionary algorithm implementations each executed 25 times on each selected problem (objective function: $z = 10v + 0.1d$, v representing the number of deployed vehicles and d representing the total distance covered by all vehicles).

implementation currently used by Athos were also unearthed in the course of the evaluation experiments: the JSprit algorithm does not only require less iteration steps to yield near optimal solutions, it's runtime performance is also superior to the one exhibited by the algorithm Athos relies upon. To this end, some performance tweaks like resorting to other data structures were applied. To further accelerate the runtime behaviour, the stratification step was modified so that it did no longer stratify the solution population but dichotomise it by only distinguishing between dominated and non-dominated solutions. However, the runtime-performance still remains something that must be improved in the near future. It must also be said that on some occasions the self-implemented algorithm produced infeasible solutions which disappeared after a few more iterations so that the executions produced a feasible near-optimal solution. The data presented in this section, however, are not affected by this. In any case, even though the Athos algorithm produced near-optimal or optimal solutions after 125 iterations, the implementation requires re-inspection and optimisation.

Sixth Chapter

Empirical evaluation of the language

This section elaborates on the empirical evaluation that was conducted to evaluate the parts of the Athos language related to the modelling of static vehicle routing problems with time windows (VRPTWs). The scope of the evaluation comprised those language elements that were presented in the first example in Section 3.4.1 (also see Section 6.7.1). The additional elements presented in the second example in Section 3.4.2 were not part of the presented evaluation and thus leave room for future work (see Section 8.3). The presented controlled experiment as well as the obtained data of the original study are to be published in (Hoffmann *et al.*, 2022).

6.1 Terminology used in this section

In this section, a terminology will be used that is in some parts slightly different from the standard terminology used in the field of software engineering experiments. Athos, as the language the described study intends to evaluate will be referred to as an *approach* instead of the term *treatment* which is generally used in such experiments (Sjoeberg *et al.*, 2005). The study was conducted among students enrolled at Technische Hochschule Mittelhessen (THM) campuses in Friedberg and Wetzlar. These students will be referred to as *language evaluators* and (far more often) as *participants* instead of the more common term *subjects*. The applied research method was a *controlled experiment* – or one argues that all participants had to apply Athos only at different points in the study (see below) then the applied

research method was a *quasi experiment* (see Sjoeberg *et al.*, 2005, p. 734). Some parts in the following discussion just use the terms *study* or *survey* instead of the more precise term 'controlled experiment'. Finally, in the general terminology the subjects apply the treatments in various *tasks*. In the presentation of this study, participants applied the approaches to answer *questions* that consisted of one or more *tasks*.

6.2 Selection of design evaluation method

In order to conduct a systematic and rigorous evaluation of the language (Hevner *et al.*, 2004, p. 80), three frameworks/templates were considered: The Usa-DSL evaluation framework[1] (Poltronieri *et al.*, 2018) was designed to assess the usability of domain-specific languages (DSLs). Based on the works of Barisic *et al.* (2012b) and Barišic *et al.* (2014), the framework regards DSLs as a user interface for human-computer interaction. The framework focuses on the usability of DSLs defined as an important part of the *quality in use* of a system. (BSI, 2011, p. 3). The Usa-DSL framework provides abstract guidelines for the evaluation process (Poltronieri *et al.*, 2021, p. 301). It does, however, not make mandatory prescriptions on how exactly specific activities are to be put into effect. Rather, the framework makes sensible suggestions on various alternatives to be taken or intentionally leaves it up to the experiment conductor, to decide on how to implement certain activities.

Several DSL evaluation studies presented by Kosar *et al.* (Kosar *et al.* (2010), Kosar *et al.* (2012) and Kosar *et al.* (2018)) served as a template to determine the concrete implementation of the abstract guidelines provided by the Usa-DSL framework. Finally, the entire research project in which Athos was developed was guided by the design-science framework presented by Hevner *et al.* (2004). While the previous chapters dealt with the *building* of innovative artefacts, this chapter focuses on the evaluation of the language which completes a first major iteration of the *build-and-evaluate loop* (Hevner *et al.*, 2004, p. 78).

[1] `https://github.com/Ildevana/Usa-DSL/wiki`

Table 6.1: Phases, steps and associated activities defined in the Usa-DSL evaluation framework (Poltronieri *et al.*, 2018).

Steps	Phases			
	Planning	Execution	Analysis	Reporting
1. Evaluators Profiles	**P1** Define evaluators profiles	**E1** Apply instruments to identify profiles	**A1** Analyse evaluator profiles	**R1** Report evaluator profiles
2. Ethical and legal responsibilities	**P2** Define informed consent form	**E2** Introduce form, collect signatures of participants		**R2** Report subjects number and form used
3. Data type	**P3** Define data type			
4. Empirical study method (SE)	**P4** Define empirical study method	**E4** Develop and conduct protocol	**A4** Analyse the development protocol	**R4** Report the development protocol
5. Evaluation method (HCI)	**P5** Define evaluation usability type	**E5** Prepare the evaluation		**R5** Report conduction evaluation
6. Metrics	**P6** Define metrics for language validation			
7. Gathering instruments	**P7** Define instruments of data gathering	**E7** Data collection	**A7** Analyse collected data	**R7** Report data analysis
8. Evaluation instructions	**P8** Def. instrum. of instruction and training	**E8** Introduce instr. of instrc. and conduct training		**R8** Report instruments
9. Evaluation conduction	**P9** Define execution place	**E9** Exec. of tasks and evaluation conduction	**A9** Analyse performed tasks	**R9** Report task analysis
10. Data packaging	**P10** Define data storage	**E10** Store data obtained		
11. Evaluation reporting	**P11** Define study reporting		**A11** Analyse documentation	**R11** Rprt results and analysed inform.

The structure of the the framework is depicted in Table 6.1: the framework divides a usability evaluation into four phases that range from the planning and execution to the analysis and reporting of the study and its results. Each phase incorporates eleven steps that (barring few exceptions) are associated with an activity to be performed in order to complete the step. As to the order in which the steps are to be carried out, it is mandatory that for a given step the order of the phases is adhered to. It is, however, not compulsory to take the steps in a specific order nor do all steps of a phase have to be completed before a subsequent phase can be entered. In general, it appears advisable to take the steps in the order implied by their numbering.

Table 6.2 displays a summary of the outcome of the activities performed in the planning phase: for example, the outcome of activity P1 *define evaluation profiles* (which is associated with the *evaluators profiles* step in the planning phase) was the decision to conduct the evaluation study among students from two different study courses offered at two different campuses of Technische Hochschule Mittelhessen (THM) in Germany. Students from the campus in Friedberg were enrolled in the *Information Systems* study course and attended a module on operations management. The curriculum stipulates that students attend this module in their second semester. Students from the campus in Wetzlar were enrolled in the *Software Technology* study course and had a module on MDSD which is stipulated to be heard in the fourth semester. The next sections will further elaborate on the activities performed in the course of the evaluation study.

6.3 Research questions and hypothesis

The works of Kosar *et al.* (2010), Kosar *et al.* (2012) and Kosar *et al.* (2018) were used as a template that provided guidance on what evaluation method to use and how to design it (Usa-framework P5). Consequently, the underlying research questions presented in these works were adapted to form the basis of the controlled experiment:

Table 6.2: Planning activities for language evaluation.

Activity	Description
P1	Students from Technische Hochschule Mittelhessen (THM). Campus Friedberg: second semester students (according to curriculum), enrolled in information systems Campus Wetzlar: fourth semester students (according to curriculum), enrolled in software engineering
P2	Consultation with the ethics committee of Technische Hochschule Mittelhessen and Napier University Edinburgh Creation of informed consent form and incorporating the terms and informations obtained from ethics committees Creation of data management plan Submission of documents and reception of permission
P3	Quantitative data (mainly), qualitative data (free text for opinion)
P4	Experimental evaluation, controlled experiment (Hevner *et al.*, 2004), comparison study in which tasks testing the learnability, perceivability and evolvability of Athos and a baseline GPL must be solved (Kosar *et al.*, 2010; Kosar *et al.*, 2012; Kosar *et al.*, 2018)
P5	Usability testing: experimental determination of effectiveness and efficiency
P6	Test score, Time required, Points per minute
P7	Online questionnaire create using Novi Survey
P8	Presentation slides and oral communication in which students are asked to solve the tasks for both approaches to the best of their knowledge an ability. No communication with fellow students and usage of IDEs. Explanation on how that would ruin the experiment.
P9	Pre-test taken by participants from research group. Presentation of learning material. Online execution of experimental study.
P10	Learning material: Storage on certified cloud drive (tuev-saar.de/TK44291 and TrustedCloud Service 10044) Raw data is only stored on servers hosted by Edinburgh Napier University and on the local machine of the thesis author
P11	Report in journal article in the journal for empirical software engineering Report in this thesis

EQ1 Does Athos facilitate comprehension of VRP models?

EQ2 Does Athos enhance efficiency in model comprehension and creation?

These research questions led to the following hypotheses:

H1 The use of Athos has a beneficial impact on the effectiveness (accuracy, correctness) of participants' results.

H2 The use of Athos significantly impacts participants' efficiency.

The next sections will provide more insight on the study population and the nature of the obtained data based on which these hypotheses are tested.

6.4 Definition of evaluators profiles: demographic information

With regard to the definition of the evaluator profiles (P1), the original intention was to conduct the evaluation study among the intended target user group of Athos, i.e. practitioners in the field of routing and traffic planning and optimisation. This however, soon turned out to entail a number of problems that were deemed to pose too great a risk for the evaluation to fail: a major obstacle was the fact that it was hard to gather a sufficient number of professionals from the field who were able to make a binding commitment on participation in a study of the intended scope. Thus, having a population of active practitioners would have implied a very small sample size limiting the statistical significance of the obtained results.

Though there were only few potential participants, it still appeared to be highly difficult to find a date (let alone multiple dates if the study was to be split over the course of several days) that would have suited every potential study participant. Thus, every single participant would have had to be trained individually on both Athos and the baseline application library that Athos was supposed to be compared to (see Section 6.6). Finally, the overall intention of the study was to evaluate the impact Athos has on users' effectiveness and efficiency. As this impact was to be evaluated by comparison to a baseline application library, study participants were required to possess some basic knowledge on the Java programming language and its underlying object-oriented paradigm. This was likely to further reduce the set of potential participants so that it was decided to invite students to participate in the study.

As was already elaborated on in Section 6.2, the study was conducted among students enrolled in *Information Systems* (Friedberg) and *Software Technology*

(Wetzlar). From the curriculum of the respective study courses it could be deduced that students from both study courses held the required minimum programming experience. It was further assumed that, on average, students from Friedberg would possess rather rudimentary knowledge in software development whereas students from the co-op *Software Technology* course where expected to be somewhat more experienced in the usage of the Java language. This is because according to the respective curriculum, the majority of students from Friedberg should be in their second semester whereas the majority of the Wetzlar students should be in their fourth semester. In addition to that, the co-op students from Wetzlar participate in industrial software-development projects in their respective company. The Friedberg group was thus deemed a suitable representation of domain experts with limited programming expertise whereas the Wetzlar group – at least to some extent – could be regarded to represent (junior) application developers.

6.5 Obtainment of ethical clearance

The second step of the Usa-Eval framework concerns the consideration of ethical and legal responsibilities of study conductors. Activity (P2) requires the definition of an informed consent form. The informed consent form presented to students who accepted the invitation to participate in the study can be found in Appendix C. A data management plan (DMP) was devised and submitted to Edinburgh Napier University. Ethical clearance was obtained from ethics committees of both Edinburgh Napier University and Technische Hochschule Mittelhessen (THM).

It is important to note that participants did not receive any reward for participation in the study. Students were informed that participation in the study could be regarded as a free training for the final exam where tasks on (modelling) VRPs were likely to occur. It was, however, emphasised that all training materials would be provided to every student independent of participation in the study. All data were anonymised upon submission. This way it was ensured that it was not possible at any time after submission to link a submitted dataset to an individual participant.

6.6 Definition of the protocol

The next activities required the definition of the type of (P3) data to collect, the empirical study method and usability evaluation type applied to obtain the data (P4), (P5) and the evaluation and the metrics to derive from the data (P6). These activities are very closely related to their counterparts in the execution phase so that these will all be discussed within this section. As regards the type of data, the focus of the study was to obtain *quantitative data* as these have been reported to be scarcely used in DSL evaluation approaches (Gabriel *et al.*, 2010; Albuquerque *et al.*, 2015). Though *quantitative data* was to take priority in the study, it was also decided to obtain *qualitative data* via free text fields that provided an opportunity for participants to express their subjective impression on both Athos and the baseline approach.

As to the study and evaluation method, the defined role-model approach of Kosar *et al.* (2012) conducted a controlled experiment in order to test the language's usability. The decision on what data were to be collected in the course of the evaluation was also based on the aforementioned works of Kosar *et al.* (Kosar *et al.*, 2010; Kosar *et al.*, 2012; Kosar *et al.*, 2018). Section 6.6.1 briefly discusses the actual data collected in the course of the controlled experiment. It also briefly discusses the language characteristics and properties (BSI, 2011, p. 2) to be evaluated based on the collected data.

6.6.1 Data to be collected and metrics to apply

6.6.1.1 Data obtained for language evaluation

According to Hevner *et al.* (2004, p. 80) the ultimate goal of any design science approach is the provision of *utility* through artefacts that meet the business needs of stakeholders. In BSI (2011), this 'degree to which the system satisfies the stated and implied needs of its various stakeholders' (p. 2) is defined as the 'quality of a system' (p. 2). The *quality in use model* defines the quality of a system in terms

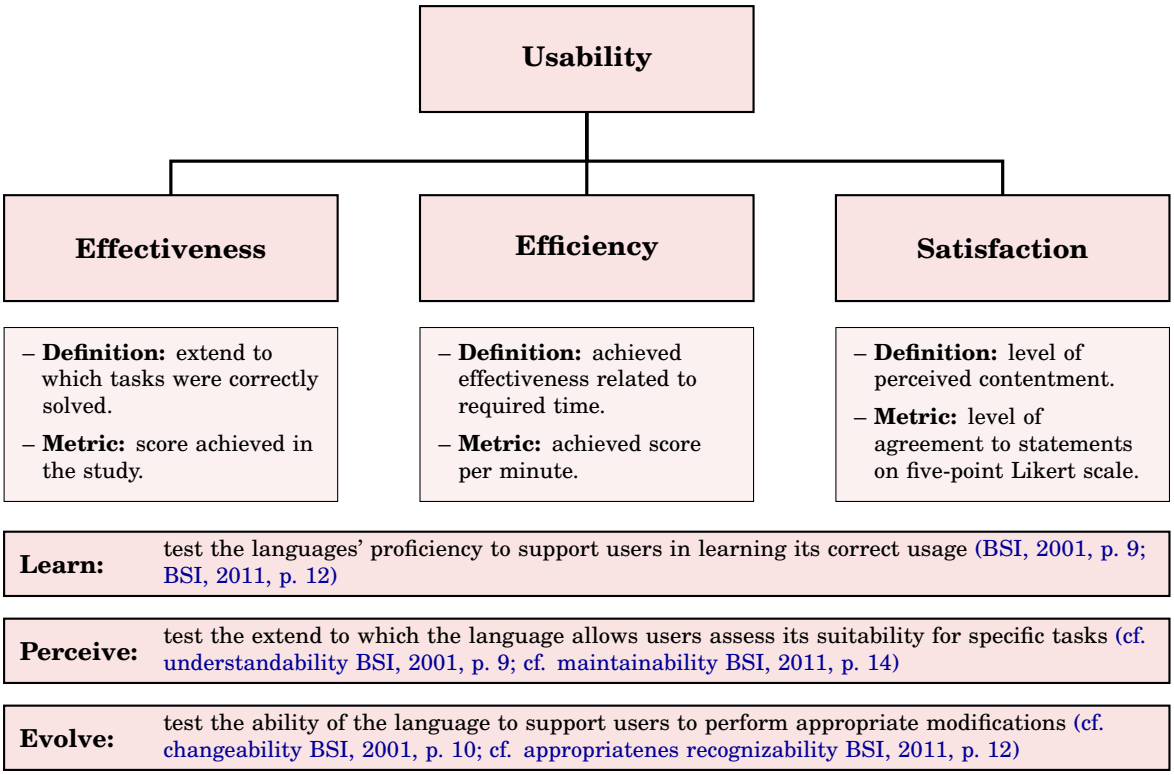


Figure 6.1: Definition of the metrics applied to evaluate the usability (BSI, 2011, pp. 8, 12) of the languages compared in the study (effectiveness, efficiency, satisfaction) together with the a set of characteristics by which the questions of the study are grouped (learnability, perceivabiliy, evolveabiliy).

of five characteristics: effectiveness, efficiency, satisfaction, freedom from risk, and context coverage (BSI, 2011, p. 3). The former three characteristics by themselves define the *usability* of a system (BSI, 2011, pp. 8, 12).

Effectiveness of a system is defined as the 'accuracy and completeness with which users achieve specified goals' (BSI, 2011, p. 8). Applied to a DSL like Athos, effectiveness thus could be interpreted as the extent to which the language supports the user in the creation of correct and complete models. According to the BSI (2011), the efficiency of a system depends on the 'resources expended in relation to the accuracy and completeness with which users achieve goals' (p. 8). Among the expedient resources is the 'time to complete the task' (BSI, 2011, p. 8). Therefore, the efficiency of a DSL can be defined as its effectiveness in the creation of models in relation to the time required to build the model. Finally, satisfaction is defined as the 'degree to which user needs are satisfied when a product or system is used in a specified context of use' (BSI, 2011, p. 8). The satisfaction achieved by a DSL could

thus be regarded as the level of contentment users experience when using the DSL to build models for the respective problem domain.

As is depicted in the upper part of Figure 6.1, the evaluation study was to provide information on the usability of Athos, i.e. on how it affects users' effectiveness, efficiency and satisfaction. These three characteristics were thus defined as the *dependent variables* of the study (see Kosar *et al.*, 2012, p. 286). Moreover, a set of questions was developed in which users where to perform one or several tasks. The complete set of these questions is presented in Appendix C. Based on the study template provided by Kosar *et al.*, the questions were further distinguished into three categories found in the *product quality model* which is also defined in (BSI, 2011, p. 2)[2]. The questions and their respective tasks aimed at three different (cognitive) activities that users engage in when working with a computer language (lower part of Figure 6.1): learn, perceive (understand) and evolve (see Kosar *et al.*, 2010, p. 251). Kosar *et al.* (2018, (p. 2741)) state that these activities are crucial in a programmers' comprehension process; to support this claim, the authors refer to a study of Hevner *et al.* (2005) that found that software developers estimated that over a quarter of their time was consumed by 'reading and understanding the behaviours of system development artefacts (e.g., specifications, architectures, designs, code, test cases) written by themselves or others' (p. 46).

In order to quantify the dependent variables, a set of metrics was defined. For a quantitative evaluation of a participants' effectiveness, points were awarded for correct answers to the respective tasks (NB: partial/custom grading was applied, so that participants would also be awarded points if an answer was not entirely correct, see Section 6.7). The total score (in relation to the maximum of achievable points) was then defined to represent the effectiveness achieved by the user when using the evaluated language. In terms of effectiveness, a metric that put the exhibited

[2]It is to be noted that Kosar *et al.* (2012, pp. 282–283) reference BSI (2001) as the source for the characteristics used in their study. Meanwhile, BSI (2001) was superseded by BSI (2011) which introduced the product quality model and also replaced the 'understandability' characteristic with the 'appropriateness recognizability' characteristic and the 'modifiability' characteristic represents both 'changeability' and 'stability' (BSI, 2011, pp. 22–23). The study in this thesis, however, still uses the characteristics suggested by Kosar *et al.* (2010, p. 251), Kosar *et al.* (2012, pp. 282–283), and Kosar *et al.* (2018, p. 2736).

effectiveness in relation to the required time was needed. In behavioural research, the rate correct score (RCS) (Woltz and Was, 2006, pp. 672–673; Vandierendonck, 2017, p. 654) metric serves exactly this purpose.

The definition of the RCS is as follows:

$$RCS = \frac{c}{\sum RT} \tag{3.1}$$

In Equation 3.1, c represents the correct answers given by a participant and the term in the denominator ($\sum RT$) represents the reaction time, i.e. the time it took the participant to complete all tasks. As is noted by Vandierendonck (2018, p. 2), the RCS is equivalent to the 'throughput' measure which 'is equal to the number of correct responses on a task, divided by the cumulative reaction times [...]' (Thorne, 2006, p. 569). If the correct answers are replaced by the total score achieved by a participant and the required time is measured in minutes and assumed to be 'discretionarily available' (Thorne, 2006, p. 570) to the participant, the RCS can be interpreted as the number of achieved points per minute[3]:

$$PPM = \frac{\sum_{i=1}^{n} S_{Q_i}}{\sum RT_{Q_i}(\text{in mins})} \tag{3.2}$$

In Equation 3.2, S_{Q_i} denotes the score achieved in the i-th question and RT_{Q_i} denotes the time in minutes that the participant spent on this question. This measure is also applied in Kosar *et al.* (2012) and Kosar *et al.* (2018). Evidently, the higher a participant's points per minute (PPM), the higher the exhibited efficiency: If participant A scored 120 points within 45 minutes (RCS: .044, PPM: 2.667) is compared to a participant B who required twice the time for the same score (RCS: .083, PPM: 1.333), then it is safe to say that A solved the tasks considerably more efficient than B did.

On the other hand, it is also important to note, that it is debatable (or dependant on the context), whether 20 points scored within just 4 minutes by a participant C (RCS: .083, PPM: 5) are to be considered more desirable than the results of A and B

[3]NB: the RCS is originally interpreted as 'the number of correct responses produced per second of activity'(Vandierendonck, 2018, p. 2).

– even though C's efficiency is superior to both A's and B's. Hence, it is important to consider both, participants' effectiveness and also their efficiency with the respective language.

To quantify users' satisfaction with a language, they were presented several statements on the evaluated language and they were asked to state their level of agreement in terms of a five-point Likert scale. The Likert scale spanned from strong disagreement and disagreement over to a neutral stance and then to agreement and strong agreement. The propositions referred to how users subjectively assessed the learnability, understandability and evolvability of the language.

6.6.1.2 Data to be obtained on the study population

In addition to the data required for the evaluation of the usability of a language, it was also essential to obtain data on the participants of the study. It is important to note here that these data were not person related and an attribution of a particular dataset to an individual participant was not possible. The sole purpose here was to gain insight on a participants' *prior knowledge* and *interest* in the field of programming.

Participants were asked for how long they had been programming and what languages they had been using so far. Participants were also asked to give a self assessment of their programming skills in terms of a five-point Likert scale that ranged from 'very poor' to 'very good'. Finally, participants were queried on their general interest in the field of programming for which another five-point Likert scale was provided that ranged from 'completely uninterested' to 'highly interested'.

The benefit of this information on the study population was two-fold: firstly, it was necessary for the verification or falsification of the assumption that the average programming experience among participants from the Friedberg study group was less than the average experience that participants from the Wetzlar study group had. This way it was possible to put the language evaluation results into perspective and detect possible effects of participants' prior experience on the measured usability of the evaluated language. Secondly, these data allowed for an assessment of the degree

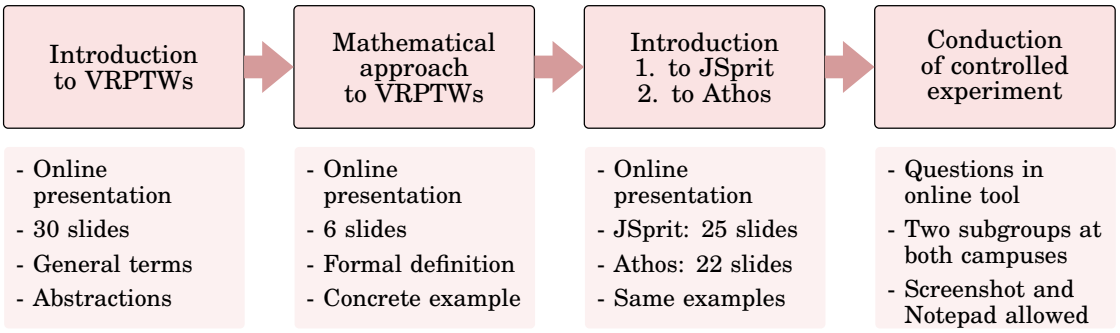


Figure 6.2: General survey conduction protocol.

to which usability results from two different study groups were comparable: in case that the average prior experience, assumed skill or general interest significantly deviated, the usability results would not be comparable without consideration of the unbalanced study groups.

6.6.2 Empirical study method and evaluation usability type

6.6.2.1 General structure

In accordance with the studies of Kosar *et al.* (2012), a controlled experiment (P4) that comprised a usability testing (P5) section for both Athos and a baseline language was conducted. The selected baseline language (see Section 6.7.1) was the application library *JSprit*[4] for the Java GPL. The results for both Athos and JSprit as a baseline approach[5] were then compared (see Section 6.6.2.2) to allow for some statistically firm propositions on the usability of Athos. These statistically backed results are crucial to demonstrate that Athos is not a mere 'routine design' but represents an IT artefact (or set thereof) that allows to address existing 'problems in more effective or efficient ways' (Hevner *et al.*, 2004, p. 81).

Figure 6.2 depicts the general structure of the evaluation study: in the upper part, the rectangles represent the performed steps in the correct chronological order. Under every step is a text box that provides some summarized additional information

[4] `https://jsprit.github.io/`

[5] A note on terminology: according to Mernik *et al.* (2005, p. 317), the combination of a GPL and a suitable application library like JSprit can also be considered a DSL. The terms 'language' and 'apporach' will thus be used interchangeably in reference to Athos and JSprit.

of relevance for the respective step. Though the initial intention was to conduct the study on the ground at the respective THM campus, this was rendered infeasible due to the emergence of the COVID-19 pandemic. Health and safety protocols enacted in the course of the pandemic required lecturing via the Zoom[6] online meeting software which was then also used for the conduction of the study.

First, participants were given a general introduction to the vehicle routing problem with time windows (VRPTW). This was done by means of a fictitious example scenario in which a grocery retailer intended to set up a home delivery service. Within this example, the objective function and the constraints of the problem were elaborated on. Participants were also introduced to the ontological concepts of the domain (e.g. depots, vehicles, products). The presentation also demonstrated how to abstract concepts like roads and storage room and transform them into mathematically processable structures.

In the second step, participants were presented a mathematical definition of the problem with a formal notation of the objective function and the time and capacity constraints. This definition was exemplified with the data from the case study. In a final step, participants were also presented a preview on how the specific vocabulary used in both JSprit and Athos in order to refer to concepts from the vehicle routing domain (e.g. JSprit uses the terms 'service', 'delivery', or 'job' for what Athos refers to as a 'customer').

After it was ensured that participants were familiar with the domain of VRPTWs, they were given an introduction to the two languages that were about to be compared in the controlled experiment. For both approaches it was shown how to apply them in order to model the example scenario used in the introduction of VRPTWs. To this end, a set of slides was prepared for each approach. It is to be noted here that utmost care was taken to design the learning material in a way that the exact same phenomena were presented in a structurally identical way, so as not to bias the study

[6] `https://zoom.us/`

results by superior introductory material (see Section 7.4) for one of two approaches (especially not in favour of Athos). Due to the different – and more verbose – concrete syntax of JSprit, however, JSprit required three more slides than Athos.

As it was believed that the first presented approach was likely to benefit from a higher level of participants' ability to concentrate, JSprit was the first approach to be introduced before Athos was presented second. This order was intended to ensure that results were, if anything, biased in favour of the baseline language. On the other hand, the second presented approach might benefit form learning effects that facilitate comprehension. The order in which the two languages were presented thus poses a possible threat to the validity of the study (see Section 7.4). Both languages were introduced by the author of this study who did his best to present both approaches with diligence and in an objective manner. However, it cannot be completely ruled out that – even if only at a subconscious level – the presentation biased the results in favour of one of the two approaches which constitutes another threat to the validity of the study (see Section 7.4).

Participants of both study groups (Friedberg and Wetzlar) were randomly assigned to one of two subgroups for the respective study (so that there were two study groups each with two subgroups which results in a total of four subgroups). In both subgroups, participants would answer questions on both languages. However, one of the subgroups would start with the set of Athos questions and then continue with the set of JSprit questions. For Friedberg, this subgroup is referred to as the *FbAf* subgroup (Friedberg, Athos first) and for the Wetzlar study the subgroup is referred to as the *WzAf* (Wetzlar, Athos first) subgroup. Both studies also featured a subgroup that was asked to answer the JSprit questions first and the Athos questions second. For Friedberg and Wetzlar these subgroups are referred to as *FbJf* (Friedberg, JSprit first) and *WzJf* (Wetzlar, JSprit first), respectively.

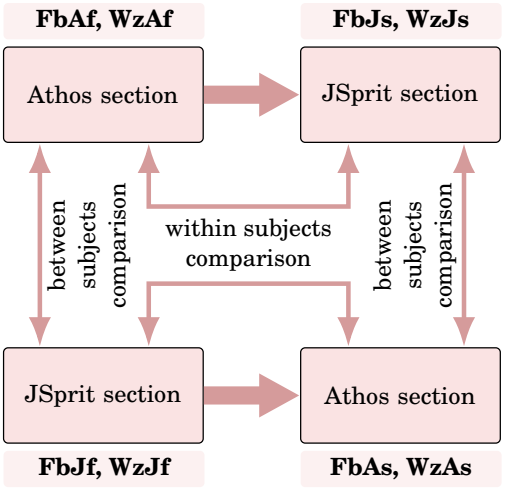
This partitioning (or subdivision) of the two study groups was essential as the questions (or tasks therein) for both Athos and JSprit were very similar (see Section 6.7.3). Hence, the likely occurrence of *learning effects* constituted another potential threat to the validity of the study as these effects were likely to bias results

in favour of the language evaluated second. By contrast, there might also occur negative effects of mental enervation or fatigue by the time a participant starts to answer the second set of questions. In terms of bias these *exhaustion effects* work diametrically opposed to possible learning effects. However, as it was not possible to reliably predict the degree to which each of these effects would occur, the decision to spilt the study groups into the aforementioned subgroups was made.

On the other hand, by the time a participant started to solve the tasks of the second approach, exhaustion effects might have started to negatively affect the performance of the participant. Participants were then given the survey URL from which they could access the survey with arbitrary browser software. After some final words of explanation regarding the voluntary basis of participation, data protection and the maximum time limit for either approach participants were asked to begin the study. Participants had two example programs at their disposal. Participants were also allowed to use Notepad++ or a comparable text editor for comparing text files. Taking screenshots was also allowed. The usage of any IDE with specific support for any of the two compared approaches was explicitly forbidden. The questions of the controlled experiment had to be answered without the help of any third person.

6.6.2.2 Statistical comparison of the obtained results

With the selected study design results can be compared in two different ways which are illustrated in Figure 6.3. Since the study groups were split into subgroups in which both approaches had to be used in different order, the results can be compared by conducting a *between-subjects comparison* and a *within-subjects comparison*. This is illustrated in Figure 6.3. As is shown, both approaches can be compared when they were used as first approaches, and when they were used as second approaches. As an original study was performed in 2020 and a replication study in 2021 the labels are also used with either an appended ‘20’ or ‘21’. This way, WzAf21 refers to the Wetzlar study subgroup of the replication study that used Athos as the first and Jsprit as the second approach.



Comparison of
- Achieved Points (Score)
- Points per minute (PPM)
- Expressed satisfaction*

Comparison of independant samples:
- FbAf vs. FbJf
- FbAs vs. FbJs
- WzAf vs. WzJf
- WzAs vs. WzJs

Comparison of dependant samples:
- FbAf vs. FbJs
- FbJf vs. FbAs
- WzAf vs. WzJs
- WzJf vs. WzAs

Figure 6.3: Different comparisons of the obtained results: results obtained from two independent samples (Athos first vs JSprit first and Athos second vs. JSprit second) as well as two dependant samples (Athos first vs JSprit second and JSprit first vs Athos second) were compared.

This perspective has the advantage, that *learning* and *exhaustion effects* were identical for both approaches when they were used by participants. The disadvantage of this way of comparing the approaches is that the results obtained by participants might be highly dependant on the individual skill level of each participant. In the worst case, the most talented participants all used one approach, and all participants generally struggling with the application of computer languages were assigned to the group using the other. This problem is addressed by comparing the data by a *within-subjects* comparison. Here the results of the same individuals are compared and so the distribution of talent among the different groups is only of reduced importance. The advantage of the between-subjects comparison is the evident disadvantage of the within-subjects comparison: learning and exhaustion effects are likely to affect the results produced by participants to a certain extend.

Statistical comparisons were performed with the IBM SPSS Statistics software Version 26.0.0.1 (64-Bit). In order to test the data for statistical significance two non-parametrical tests were used. To test the within-subjects data for statistical significance the Mann-Whitney U test (McKnight and Najab, 2010) for independent samples was applied. For the within-subjects comparison the Wilcoxon singed rank test (Woolson, 2007) for dependent samples was used.

6.7 Definition of instruments to obtain the data

The data were obtained with the NoviSurvey online survey tool. The tool was used to define a set of questions that gave insight on a participant's prior programming knowledge and interest in the field of programming. Two sections with questions for both compared approaches were defined with the NoviSurvey tool. These sections contained questions in which participants had to check the correct check boxed in order to answer the question, or select the correct option button. There were also questions with free text fields in which participants had to enter program code.

The obtained data was processed in Excel. For multiple-choice answers a VBA script was written to assign points (or partial points) depending on the radio-buttons selected or the number of correctly ticked checkboxes. The questions together with the correct answers can be found in Appendix C.

6.7.1 Selection of baseline language

The template study of Kosar *et al.* (2010) applied a comparative approach that compared the usability of a DSL to the usability of an application library used within a GPL. Hence, a suitable application library was to be selected and compared to Athos. However, there was no language available that had an identical problem domain and at the same time was appropriate for the intended participants of the study (e.g. XML-based modelling approaches were not regarded as suitable baseline languages to be applied by students in their second semester). For this reason it was decided, that the baseline approach did not necessarily have to be exactly equivalent to Athos in terms of its capabilities as long as it allowed for the modelling of vehicle-routing related problems.

The JSprit[7] application library was hence chosen as the most suitable baseline approach for the evaluation study. JSprit was found to be an appropriate baseline approach for two reasons: first, its underlying problem domain is similar to the

[7] `https://github.com/graphhopper/jsprit`

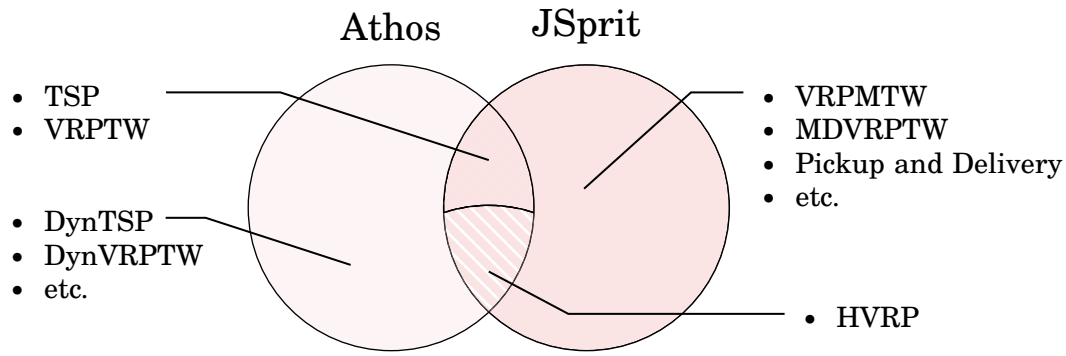


Figure 6.4: Comparison of the capabilities of Athos and JSprit.

problem domain of Athos; second, as an application library for the Java language it would not confound the study results with an underlying language that is alien to the study participants. On the contrary, students of both study groups were supposed to have prior experience with the Java language so that comprehension of a Java-based API was a reasonable task to master.

Though the targeted domains of both Athos and JSprit are similar as both allow the modelling of static vehicle-routing problems, they are not identical. With the modelling of routing problems, both approaches have a similar problem domain at their core. However, both approaches were defined for different purposes: Athos is a DSL for modelling *various* different traffic and vehicle-routing *scenarios* in which optimisation problems assume an important role. By contrast, JSprit is an application library entirely focused on static vehicle-routing problems. Therefore, Athos is capable of modelling dynamic aspects such as congestion factors. JSprit, on the other hand, has a very broad range of different routing problems it can model and solve. For this reason, it was important to confine the study tasks in a way that they were covered by both approaches.

The Venn-Diagram in Figure 6.4 compares the features offered by Athos and JSprit. Athos can be applied for modelling several static VRPs like the TSP or VRPTWs. The generated simulation also allow insight into dynamic versions of these problems (see Pillac *et al.*, 2013, pp. 2–3), as those agents that calculated an optimal solution can be affected by other agents that may slow them down and thus alter the problem so that a recalculation of the optimal tour might become necessary.

JSprit does not offer these dynamic capabilities. However, it allows for modelling various problems (e.g. pick-up and delivery problems) that currently cannot be modelled in Athos directly.

Though JSprit supports modelling of several problems, that are not directly supported by Athos, Athos offers an *extension* mechanism for the introduction of additional information into an Athos model and map these to an annotated Java algorithm. In Hoffmann *et al.*, 2019a this feature was presented by adding additional information to an Athos model so that a *heterogeneous vehicle-routing problem with vehicle type dependant routing costs* could be modelled with Athos. For this reason, the lower part of the intersection shown in Figure 6.4 represents problems that can be mapped via this extension mechanism. The reason that this mechanism is not discussed in more detail in this thesis is that its application yields models that are considerably more difficult to comprehend than Athos programs which solely rely on built-in mechanisms. Though it is a mechanism that offers *user extensions* (Atkinson and Kuhne, 2003, p. 37) , additional research is required to ensure that extensibility does not come at the cost of usability (see Section 8.3.2).

6.7.2 Definition training material

Before participants took part in the actual controlled experiment, they received training for both approaches. To provide a general undrestanding for VRPTW, a set of slides with general information on the topic was created. This set consisted of 36 slides that presented a fictitious sample case in which a VRPTW was to be solved. The slides present the process of abstraction from the real-world problem to the formulation of a mathematical model.

For each of the two approaches a set of slides was created. Both sets explained how the problem presented in the general set of slides was to be modelled with the respective approach. Utmost cate was taken to present the exact same information in both sets as to not bias the results through different training material. The set of Athos slides featured 22 slides, the set of JSprit slides contained 25 slides since

some programs were more verbose in JSprit. For both approaches, participants received two cheat-sheets that showed the example program iteratively created in the slides as a whole as well as a variation of the program that was also addressed in the slides.

6.7.3 Definition of the survey tasks

The survey tasks were centred around different questions on VRPTW and different elements of the approaches used to model VRPTW. The respective topic of the question or intention of the question was based on the question presented in the template study of Kosar *et al.* (2010). The questions are provided in Appendix C.

It was highly important to define the questions in a way that made it most unlikely that study results were biased by the design of the questions. Utmost care was taken to not make the tasks for one approach more complex than for the other. In order to do so, corresponding questions were always *structurally similar*. This means that the tasks in two corresponding questions featured the same number of elements, i.e. the same number of depots, nodes, edges and agents. While this mitigates a threat of validity in terms of different levels of difficulty of the questions, it increases the threat of validity due to learning effects (see Section 7.4).

6.7.4 Inclusion and exclusion criteria

The definition of inclusion and exclusion criteria was not a trivial task as it was required to strike a balance between a larger population study that allows a greater statistical representativeness and the prevention of biases through inclusion of cases that were not representative. However, this raises the question to when exactly a case is to be considered as a representative case. This was especially difficult to decide with regard to the allowed number of unanswered questions before a case was excluded from the study population. Another difficult question was at what point a case was to be considered an outlier and thus to be removed from the study.

While the decisions made in regard to these questions may always be debated, they were at clearly defined and consistently applied.

A dataset was to be removed from the population study, if

- in sum more than seven questions were
  - opted-out of by selecting the n/a button provided for every question,
  - no button was clicked at all (would also apply if not answered due to 90-minute cut-off),
  - a text area was left blank,
  - a text area contained text undoubtedly not intended to solve the task.
- the difference of time spent on answering the questions of the two approaches were detected as an outlier in a stem-and-leaf analysis conducted with the statistical analysis software. This analysis was performed within each subgroup (FbAf, FbJf, WzAf, and WzJf).
- the difference of the obtained points for both approaches was detected as an outlier in a stem-and leaf analysis conducted with the statistical analysis software. This analysis was performed within each subgroup (FbAf, FbJf, WzAf, and WzJf).

6.7.5 Study protocols

6.7.5.1 Original evaluation study

Figure 6.5 illustrates the study protocol of the studies conducted with the Friedberg and Wetzlar. In Friedberg 118 students took part in the study (i.e. submitted a data set for evaluation). 63 of the participants that submitted their answers were members of the Athos first subgroup, 55 were assigned to the JSprit first group. The study in Friedberg was conducted over a time span of eight days. There were four days at which Zoom meetings took place. On the 17th of June participants were introduced to the topic of vehicle-routing problems by means of a fictitious

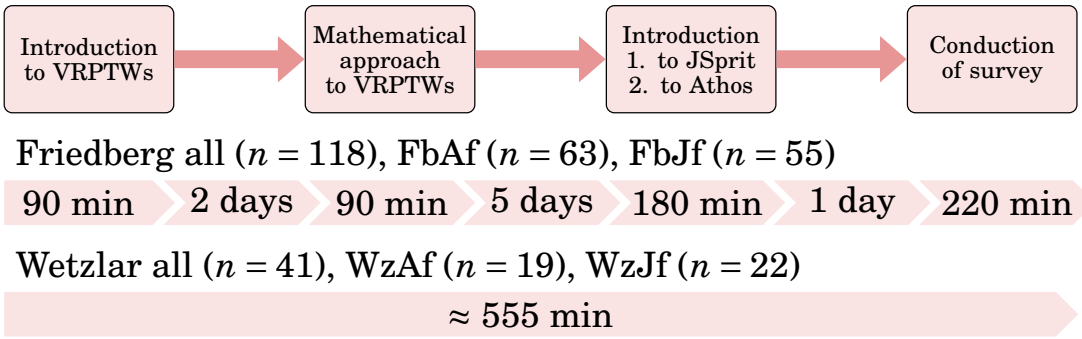


Figure 6.5: Protocol for the controlled experiment conducted in 2020.

scenario that featured several real-world aspects that were abstracted and made mathematically accessible. Two days later, the second part of the introduction was conducted in which the mathematical definition presented in Section 3.2.3 was elaborated on. Both of the introductory sessions were performed within a 90 minute time frame. On the 24th of June, participants were introduced to both approaches. The introduction started with JSprit and then Athos was introduced second. One day later, on the 25th of June, the controlled experiment was conducted within a time frame of 220 minutes.

Due to a very strict time table of participants from Wetzlar, the protocol with these students had to be different. Here, introduction of the general topic, presentation of both approaches, as well as conduction of the controlled experiment all took place on the 9th of July, 2020 within a time frame of ≈ 555 minutes.

6.7.5.2 Replication study

In order to validate the results from the original controlled experiment, a replication of the controlled experiment was conducted. As in the year before, students were enrolled at THM campuses in Friedberg and Wetzlar. The study with the Friedberg students was again conducted over several days while the Wetzlar study again was completed in a single day.

The introduction of participants from Friedberg to the topic of VRPs, the presentation and introduction of Athos and JSprit, and the execution of the controlled experiment took place between 23rd of April and 7th of May. For the 2021 Wetzlar

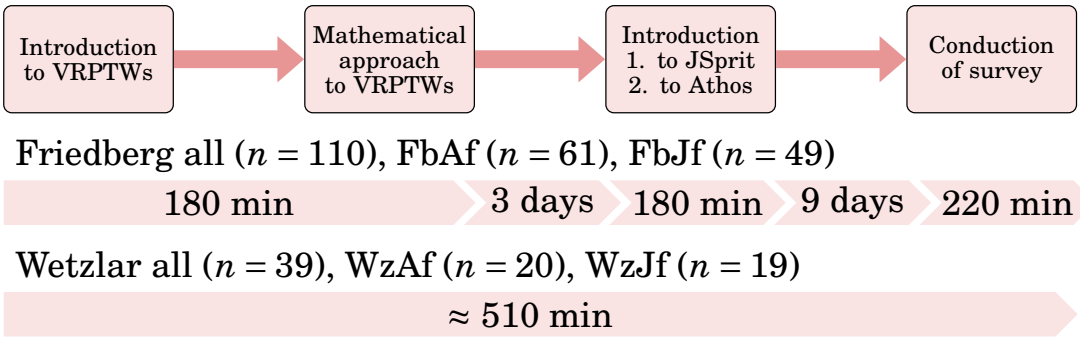


Figure 6.6: Protocol for the controlled experiment conducted in 2021.

study group, the experiment was conducted on the 15th of July. The times required for the respective parts of the study (and the days between events) are shown in Figure 6.6. Most participants participated in the controlled experiment for the first time, though some already were among the participants from the original study (see Section 7.4).

Seventh Chapter

Evaluation results

This section presents the results of the empirical language evaluation. Section 7.1.1 discusses the results obtained in the original evaluation study conducted among students from Friedberg and Wetzlar in 2020. Section 7.2 presents the results obtained in the replication study conducted among students from the same campuses in 2021.

Both sections follow the same pattern in the presentation of the results. They start out with a presentation of the overall numbers of obtained cases and the number of included and excluded cases after application of the pre-defined criteria (Section 7.1.1 and Section 7.2.1). After that both sections discuss the demographic information obtained from the included cases (Section 7.1.2 and Section 7.2.2). Subsequently the points achieved with both approaches will be presented in order to discuss the results in terms of observed correctness (Section 7.1.3 and Section 7.2.3). Next, these results will be discussed in light of the temporal dimension to gain insight on the observed efficiency (Section 7.1.4 and Section 7.2.4). The presentation of the results is concluded by a presentation of the results on observed user satisfaction levels (Section 7.1.5 and Section 7.2.5).

7.1 Results of the original study

This section will present the results of the empirical language evaluations. It will follow the structure presented at the beginning of the chapter. The aforementioned

sections are always split into a subsection for the data obtained from Friedberg and a subsection for the data obtained from Wetzlar. For example, Section 7.1.2.1 first presents the demographics data of participants from Friedberg, and subsequently Section 7.1.2.2 presents demographic data on participants from the THM campus in Wetzlar.

7.1.1 Application of exclusion and inclusion criteria

Table 7.1: Number of cases excluded from and included in the analysis of the 2020 study.

		Non-attempt filter						
Subgroup	Pre	#NA_AT	#NA_JS	OR*	AND	OL_TME	OL_SCR	Included
FbAf20	63	16	23	24	15	4	0	35
FbJf20	55	16	14	17	13	0	0	38
WzAf20	19	1	2	2	1	0	2	15
WzJf20	22	1	0	1	0	1	2	18

Table 7.1 reports the number of cases originally obtained from each subgroup in Friedberg and Wetzlar together with the number of cases that were removed by the applied filters that were applied. The first column indicates the subgroup for which the numbers are reported. The second column (prefiltered) indicates the number of cases submitted via the online survey tool. The #NA_AT column indicates the number of cases with more than seven non-attempts for Athos, and the column #NA_JS indicates the number of cases with more than seven non-attempts for JSprit. The OR column states how many cases had more than seven non attempts for either of the two approaches and were thus removed from the study. The AND column informs on how many data sets had more than seven non-attempts for both approaches[1] (cases with more than seven non-attempts for either Athos or JSprit). OL_TME displays the number of cases that were removed due to a significant

[1]This column is presented to allow for further thoughts on whether Athos or JSprit might be the reason for participants to opt for not answering more than seven questions.

divergence in the time spent for one of the two approaches. OL_SCR reports on the number of cases that were excluded due to an unreasonably high difference in the score achieved with the two compared approaches.

As can be seen from the reported data, in Friedberg there were considerably more cases removed because participants did not provide answers for more than seven questions. This was to be expected as the study course in Friedberg is far less organised like a school in comparison to the study course in Wetzlar. In Wetzlar, students are subject to compulsory attendance so that it is likely that they felt that they had to take part in the study and answer all the questions despite it was mentioned on numerous occasions that participation in the study was completely voluntary.

The data presented in the table also shows that the vast majority of cases were removed by the first filter. After the application of the non-attempt filter, the remaining filters only removed a small number of additional cases. In The Athos first group in Friedberg, for example, 4 cases were removed as participants spent considerably more time on answering the questions for one of the two approaches.

7.1.2 Demographic data

7.1.2.1 Demographic data Friedberg 2020

Figure 7.1 presents the results of questions that inquired on participants' prior programming knowledge. The bar charts compare the results from the two sub-groups of the Friedberg 2020 study group. The questions asked participants on the number of years they had been programming before participating in the study, the number of programming languages they had used so far, how participants would assess their own programming skills and how interested participants were in the topic of programming in general.

As can be seen from the bar charts, both groups had a comparable overall experience and interest in the topic of programming. As concerns the number of years participants had been using programming languages, the bar charts show that

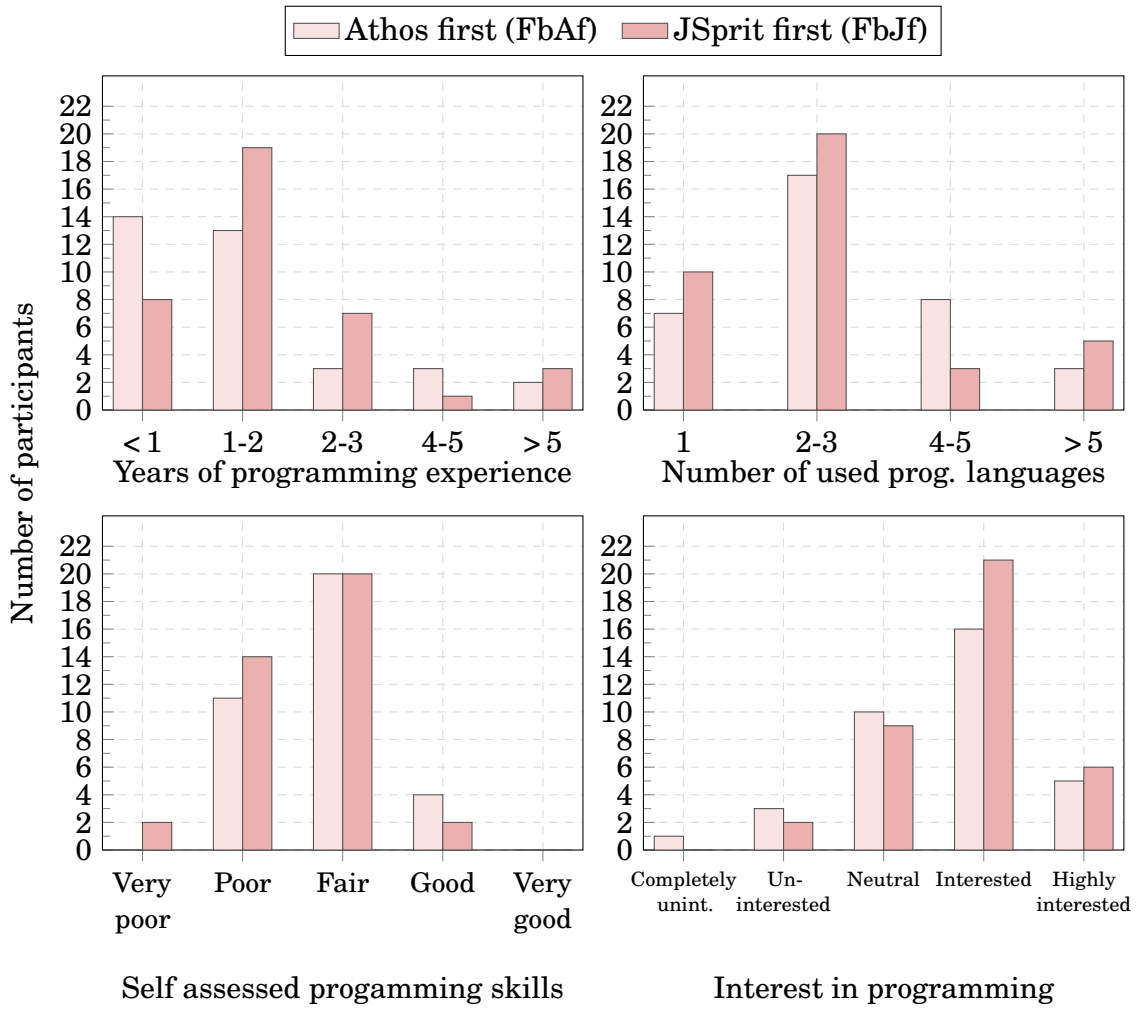


Figure 7.1: Comparison of participants' prior programming experience, number of programming languages used, self-assessed programming skills and general programming interest between the Friedberg 2020 study group using Athos first (▭) and the Friedberg 2020 study group using JSprit first (▭).

participants in both groups were mostly beginners in the usage of programming languages. Of the 35 participants from the Athos first subgroup, 27 had less than two years of programming experience (≈ 77 %). In the JSprit first group, 27 of all 38 participants (≈ 71 %) had not been programmers for more than two years. Both groups only had a small proportion of participants that had already been programming for more than four years with 5 participants (≈ 14.2 %) from the Athos and 4 participants from the JSprit group (≈ 10.5 %) falling into this category. There was a slightly larger number for JSprit in the group of participants with an experience of around two to three years, but overall it can be said that the groups were well balanced with respect to the number of years as a programmer.

As to the number of used programming languages both groups mostly consisted of participants that had not used more than 3 different computer languages[2]. In the Athos group, 24 participants belonged to that category (≈ 69 %) and in the JSprit group 30 participants from that category were present (≈ 79 %). Participants from both groups gave a rather reluctant self-assessment of their own programming skills. While in both groups most participants considered themselves to be 'fair' programmers, both groups also feature a substantial number of participants that regarded their own skills as rather insufficient with a self-assessment of 'poor' or even 'very poor' (Athos ≈ 31 %, JSprit ≈ 42 %). Though participants in the JSprit group were slightly less confident, both groups can still be regarded as balanced in that matter.

The majority of participants from both groups stated to be interested in the topic of programming (Athos ≈ 60 %, JSprit ≈ 71 %). With 4 participants compared two 2, there were slightly more participants in the Athos group who stated their disinterest in the topic of programming. One participant even went so far as to express a complete disinterest in the topic. Hence, there is a slight advantage with regard to interest for the group that applied JSprit first. However, the advantage is still within an acceptable scale so that the groups do not have to be regarded as skewed in this respect.

7.1.2.2 Demographic data Wetzlar 2020

The results of the programming background questions obtained from the Wetzlar study group in 2020 are illustrated in Figure 7.2. From the bar chart in the left upper corner it can be seen that – in stark contrast to the Friedberg study groups – both subgroups in Wetzlar only had a small number of participants that had only been programming for a maximum of two years. In the Athos first group two participants had a beginner's number of years (≈ 13 %); in the JSprit first group 4 participants had less than two years of programming experience (≈ 20 %) with one participant

[2]It has to be noted here that some participants considered mark-up languages such as HTML as programming languages.

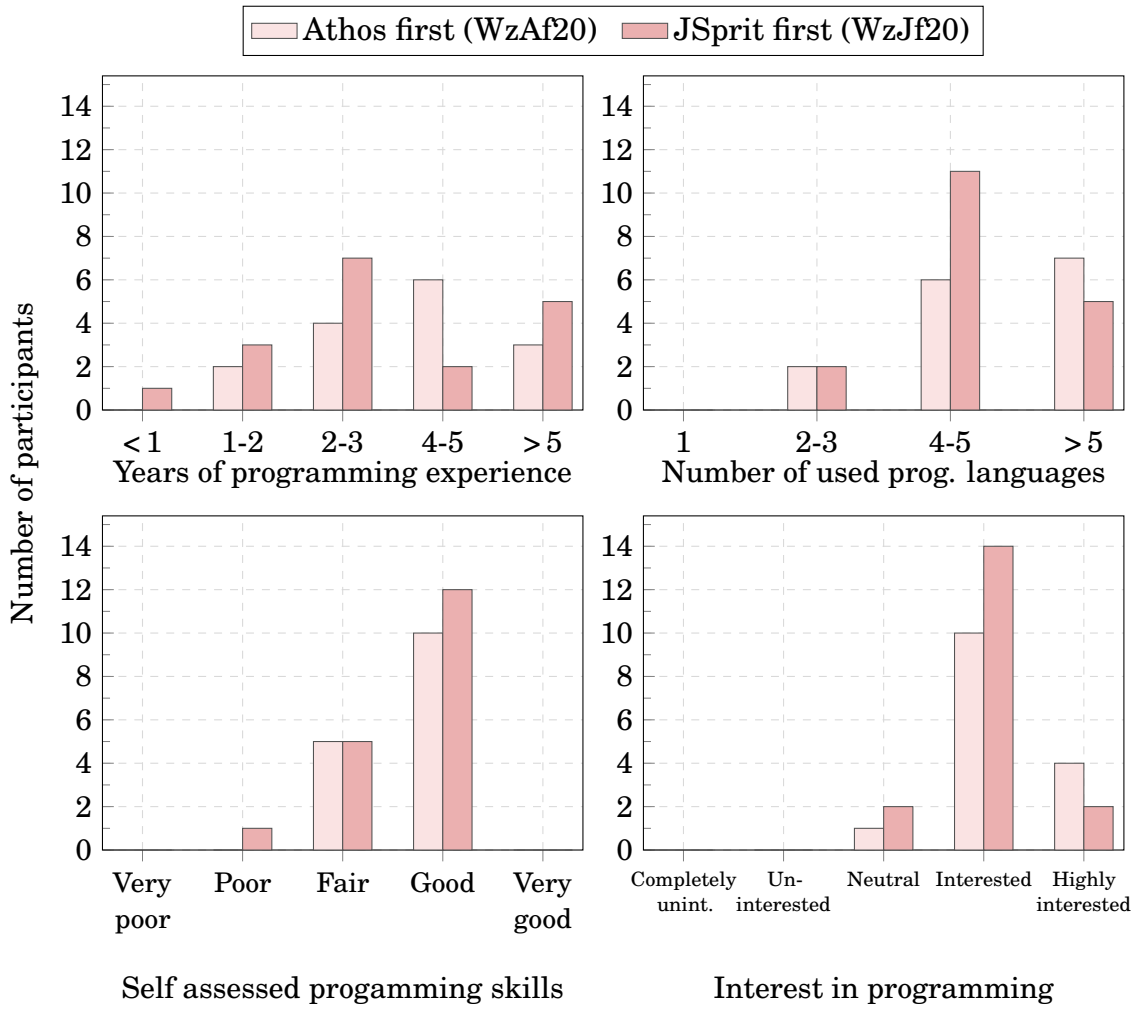


Figure 7.2: Comparison of participants' prior programming experience, number of programming languages used, self-assessed programming skills and general programming interest between the Wetzlar 2020 study group using Athos first ( ) and the Wetzlar 2020 study group using JSprit first ( ).

not even having one year of experience. Though slightly differently distributed, both groups were also balance with regard to moderately experienced to well-experienced programmers: the Athos group had 4 participants with 2 to 3 years of experience (≈ 27%) whereas the JSprit group had 7 participants in this category (≈ 39%). The Athos group had a considerably higher percentage of participants with 4 to 5 year of experience (40 % vs. ≈11 %) but in the JSprit group there were substantially more participants with a programming experience of more than 5 years (20 % for Athos compared to ≈28% for JSprit). In total, this dispersion in programming knowledge appears to be acceptable though the Athos group might have a slight advantage here.

With regard to the number of programming languages, both groups mainly consisted of participants that had used four or more different languages. In the Athos group 13 participants had some experience with four or more languages (≈87 %), in the JSprit group 16 participants had applied this number of languages (80 %). Also, the vast majority of participants from both group considered themselves to be at least 'fair' or even 'good' programmers (Athos 100 %, JSprit ≈ 94%), though no participant claimed to be a 'very good' programmer. Finally, both groups consisted almost entirely of participants that were at least interested in programming (Athos ≈93%, JSprit ≈ 80 %.

Overall, there was no obvious skewness with regard to prior knowledge among participants of both subgroups. Comparing the study groups from Friedberg and Wetzlar, however, the expected difference in terms of prior knowledge can be observed from the obtained data. In every single category, participants from the Wetzlar study group were (on average) superior to participants from Friedberg. This means participants from the Friedberg can be considered to represent language users with limited programming experience such as domain experts in transport optimisation. Participants from the Wetzlar group appear to be suitable to represent the software developers in a development project who know various programming languages and paradigms and have a considerable interest and experience in the application of programming languages.

7.1.3 Results in terms of correctness

7.1.3.1 Results from Friedberg

The box-and-whisker plot in Figure 7.3 illustrates the results of the Friedberg study group in terms of achieved correctness. A between-subjects comparison of the scores both groups achieved with their respective first approach shows that the median value in the Athos first group was a substantial 38 points higher than the median score observed in the JSprit first group. The median score in the Athos group was also 2.7 points above the upper quartile found for the JSprit group. The lower

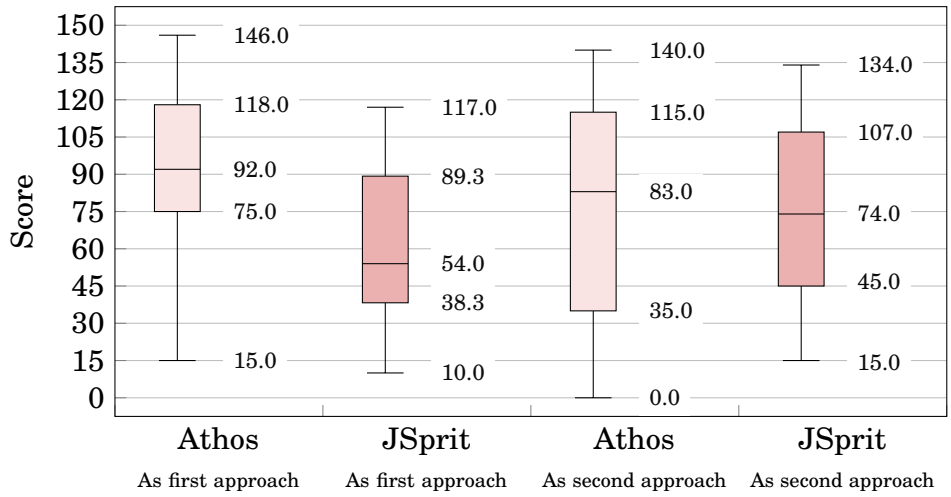


Figure 7.3: Boxplot showing the distribution of the scores achieved by participants of the study conducted in Friedberg in 2020 ($n_{\text{FbAf20}} = 35, n_{\text{FbJf20}} = 38$).

quartile for Athos was 75.0 and thus a marked 21.0 points above the median score from the JSprit first group. Though both subgroups produced similar minimum scores, the highest score achieved with Athos was substantially higher than the highest score in the JSprit group. Finally, it is to be noted that the upper quartile observed in the Athos group is also one point above the maximum score for the JSprit group.

A between subjects comparison of the results when the approaches were used second shows that the median score with Athos was 9.0 points above the median score achieved with JSprit. The maximum score with Athos was also 6 points above the maximum score in the JSprit group. However, the lower quartile in the JSprit group was at 45.0 points and thus 10 points above the lower quartile of the Athos group. The interquartile range with Athos was 80 points and thus the scores in this group were more dispersed than those in the JSprit group where the interquartile range was 62.0 points.

A within subjects comparison for the Friedberg group that used Athos first and JSprit second also shows favourable results for Athos. With Athos, the maximum score was a considerable 12 points above the maximum score achieved with JSprit as the second approach. The median dropped by 18 points from 92 points with Athos first to 74 with JSprit second. The lower quartile for Athos first still is one point

above the median observed for JSprit second. Both approaches had an identical minimum score of 15 points.

A within subjects comparison among the group that used JSprit as their first and Athos as their second approach shows that the observed results improved with Athos. The median value increased by as many as 29.0 points from a mere 54.0 points to a respectable 83.0 points. The maximum score with JSprit first was 117.0 points and showed a substantial increase of 23.0 points when Athos was applied second. The interquartile range also increased from 51 points with JSprit as a first approach to 80 points with Athos used second.

In general, the observed scores are in favour of Athos. Not only do both between-subjects comparison show that participants produced a higher average score with Athos, but it can also be observed that the achieved points decrease when Athos is used first and JSprit is used second. On the other hand, the average points achieved increase when JSprit is used as a first and Athos is used as a second approach. In the former case, JSprit does not appear to benefit from possible learning effects when used as a second approach. In the latter case, the application of Athos as a second approach ensues in a considerable increase of the average score that may partly be explained by the occurrence of learning effects.

7.1.3.2 Results from Wetzlar

Figure 7.4 shows the study results produced by the subgroups of the Wetzlar study. The first thing to notice is that throughout all datasets the data were considerably less dispersed than the data from the Friedberg group. With the exception of a few outliers (see Section 7.4), most of the observed scores were within close distance which results in small interquartile ranges in all datasets.

Comparing the results for both approaches used first, the group that used JSprit produced better results than the Athos group. The median for the JSprit group was 126.0 points which is 18.0 points above the median of the Athos group and in close range to the upper quartile observed in the Athos group. The Athos group also had a

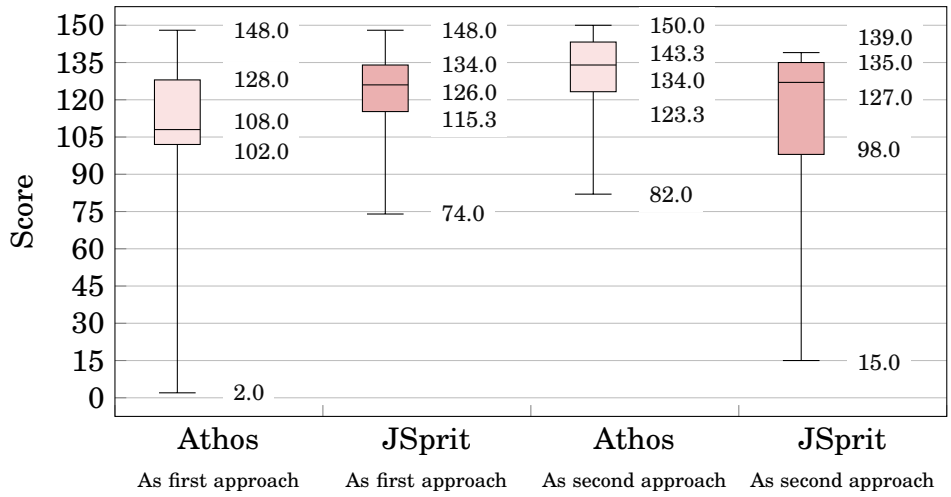


Figure 7.4: Boxplot showing the distribution of the scores achieved by participants of the study conducted in Wetzlar in 2020 ($n_{\text{FbAf20}} = 15, n_{\text{FbJf20}} = 18$).

significant outlier which places the lower whisker for the group at a very poor score of 2.0 points.

When Athos and JSprit were used as second approaches, results are almost diametrically different to when both approaches were applied as first approaches. The median in the Athos group now was 7.0 points above the median in the JSprit group and the lower quartile in the Athos group was 25.3 points above the one observed in the JSprit group. The upper whisker in the Athos second plot shows that there was at least one participant who scored a perfect 150 points which was 11 points above the best score found among the JSprit second participants. These results may be considered as an indicator that participants of the JSprit first group were generally more effective and efficient independent of the applied approach.

A within subjects comparison of the results produced by the Athos first group shows that participants from that group performed better with JSprit as their second approach. The median with Athos first increased from 108 to 127 points when JSprit was used second. On the other hand, the maximum score declined from 148.0 points to 139.0 with JSprit applied secondly. In the other group that started with the JSprit questions and answered the Athos questions thereafter, there was also an increase in the average number of achieved points. With JSprit first, the median was at 126.0 points and it was raised by 8 points to 134 poitns with Athos as a second approach.

From these data no definite conclusion can be drawn. Some of the presented comparisons are in favour of JSprit, whereas others suggest Athos as the superior approach. It is highly likely that there were additional factors other than the applied language which had a substantial effect on the final results.

7.1.3.3 Between subjects significance tests on observed correctness

To test the obtained results for statistical significance the Mann-Whithey U test was applied. The results of the tests are shown in Table 7.4. The first line of the table shows that participants in Friedberg scored a significantly higher number of points (Mdn 92) with Athos then they did with JSprit (Mdn 54). With Athos as a second approach, participants also scored a higher number of points when using Athos (Mdn 83) then they did with JSprit (Mdn 74), however, this was not statistically significant.

On a surprising note, participants in Wetzlar achieved a higher number of points with JSprit as the first approach (Mdn 126) then they did with Athos (Mdn 108). However, this results was also not statistically significant. As a second approach, the scores achieved with Athos (Mdn 134) were higher than those achieved with JSprit (Mdn 127) though also not statistically significant.

Though only one of the four tests was statistically significant, the results from the Friedberg study group provide evidence that Athos can enhance the correctness of users with little knowledge in the application of programming languages. The results of two other compared data set were also in favour of Athos, though they

Table 7.2: Between subjects comparison of the achieved scores in 2020 studies using the Mann-Whitney U test.

	N		Mdn_{Score}		Mdn_{Rnk}				
	Athos	JSprit	Athos	JSprit	Athos	JSprit	U	Z	p
Fb as first	35	38	92	54	46.130	28.590	345.5	−3.529	**.000**
Fb as second	38	35	83	74	38.040	35.870	625.5	−0.436	.663
Wz as first	15	18	108	126	14.300	19.250	214.5	−1.468	.142
Wz as second	18	15	134	127	19.220	14.330	95.0	−1.448	.148

were not statistically significant. Since participants from Wetzlar had been using programming languages like Java for a longer period of time than participants from Friedberg, it was likely that they would produce somewhat better results with JSprit. However, the observed difference in the medians was still somewhat surprising.

7.1.3.4 Within subjects siginificance tests on observed correctness

The results of the statistical significance tests that testet the within-subjects perspective for statistical significance are reported in Table 7.3. In the Friedberg study the scores achieved were significantly higher with Athos as the first (Mdn 92.00) than with JSprit as the second approach (Mdn 74). Participants also scored significantly higher with Athos as their second approach (Mdn. 83) than they did with JSprit used first (Mdn. 54).

In Wetzlar, participants scored higher with JSprit second (Mdn 127) then with Athos first (Mdn 108). However, this result was not statistically significant. When JSprit was used first (Mdn 126) participants scored significantly higher with Athos as the second approach (Mdn 134).

With three out of four within-subjects comparisons providing statistically significant evidence for a higher score achieved when using Athos, it is reasonable to claim that the data from the study indicate that Athos has the potential to enhance users' correctness in the process of comprehending and creating models for vehicle routing problems with time windows (VRPTWs). The group that achieved better

Table 7.3: Within subjects comparison of the scores achieved in the 2020 studies using the Wilcoxon signed-rank test.

			Mdn_{Score}				
	n	Ties	Athos	JSprit	W	Z	p
Fb Athos first	35	0	92.0	74.0	84.5	-3.776^a	**.000**
Fb JSprit first	38	1	83.0	54.0	135.0	-3.267^a	**.001**
Wz Athos first	15	0	108.0	127.0	38.5	-1.223^b	.221
Wz JSprit first	18	1	134.0	126.0	19.5	-2.701^a	**.007**

[a] Based on positive ranks

[b] Based on negative ranks

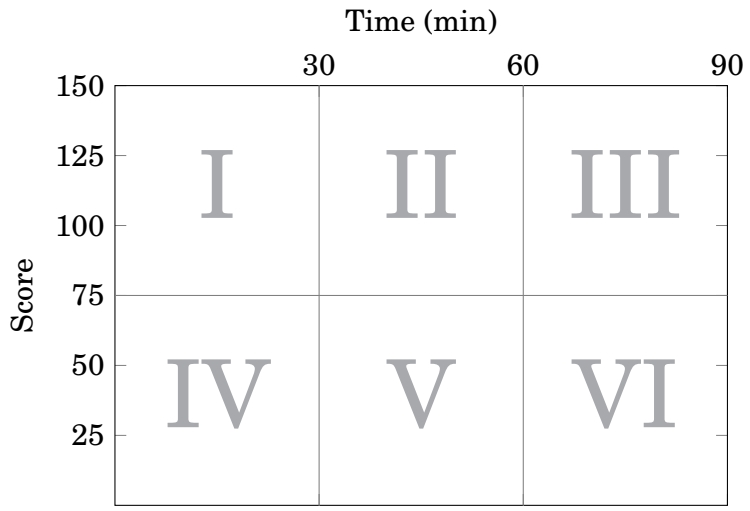


Figure 7.5: General naming schema applied to the sectors of the scatter plot.

scores with JSprit as the second approach than with Athos first, however, can be interpreted in a way that suggests that the improved correctness is also dependent on the prior programming experience of the users. With programmers that are used to a programming language, the positive effects potentially brought about by Athos might be overruled by effects of being used to a certain programming style and potential learning effects that likely occur when performing a similar task for a second time. Nevertheless, the presented data gives reason to believe that the choosing Athos for modelling vehicle-routing related problems is a worthwhile decision.

7.1.4 Results in terms of efficiency

This section will present the results from the studies conducted in Friedberg and Wetzlar with the addition of the temporal dimension. For the visualising the achieved scores of participants in relation to time scatter plots similar to the one shown in Figure 7.5 will be used. The x-axis displays the time participants spent on answering the questions on (or solving the tasks with) the respective approach. The x-axis is cut off at the 90 minutes mark since the implemented survey timer would not allow any participant to spent more than 90 minutes on answering the questions of either of the two approaches. The axis is divided into three sections so that three intervals $[0,30),[30,60),[60,90]$ ensue. It is reasonable to assume that the lower the amount of time required by a participant to obtain a particular number of points, the higher

their efficiency in obtaining that score, in other words, it is desirable for an approach to have as many data points as possible on the lower end of the x-axis.

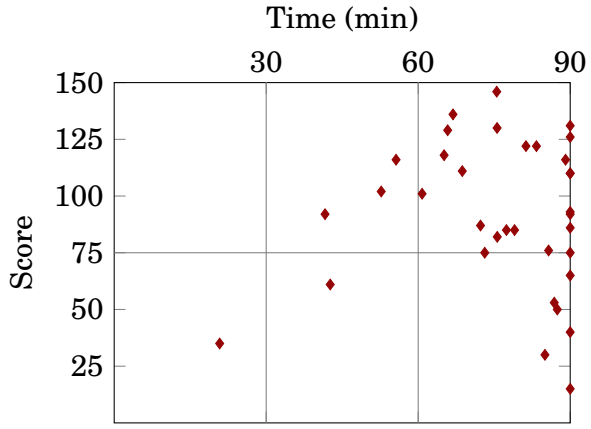
The y-axis displays the number of points participants were awarded for their answers. The maximum number of points to be awarded was 150. The y-axis is split at the 75 points mark. This was done because it is common practice to devise marking schemes that have participants pass an exam with at least 50 per cent of the maximum score. However, it is important to note that since all the results obtained from participants were anonymised, participants could actually neither pass nor fail the in the controlled experiment and so they were not given any instruction that any minimum score was required. For the sake of brevity, however, the discussions in the next sections will refer to participants with at least 75 points as passing participants whereas participants with less than 75 points will be said to have failed the respective section.

Together, the lines used to part the x- and y-axis form six sectors (or sextants). At some paragraphs the number shown in the illustration will be used to refer to a particular sector. The presented counting schema was chosen as it reflects the usual reading direction from left to right and top to bottom. However, a case could also be made for that the numbers reflect on the desirability for a data point to be plotted in the respective sector with I being the most and VI the least desirable[3]. Especially for the study in Friedberg, comprehension and mental aggregation of the plotted points can become rather difficult. For this reason, the illustrations in the following sections will also present a table with the aggregated number of data points in each sector.

7.1.4.1 Results from Friedberg with first approach

Figure 7.6 displays the results participants in Friedberg achieved with Athos and JSprit as a first approach. With JSprit, 5 participants achieved considerably less than 25 per cent of all points while spending less than 30 minutes. With Athos as

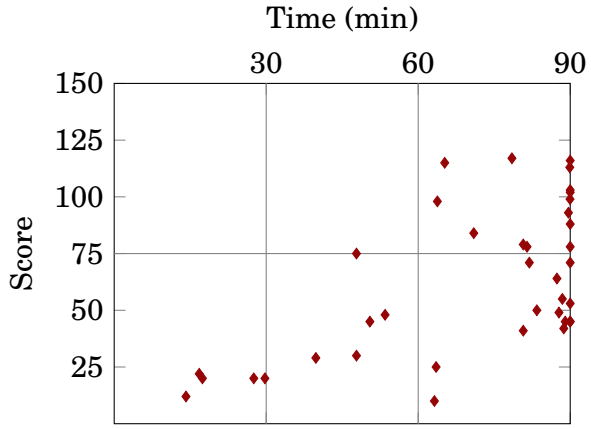
[3]Though sectors IV to VI can all comprise undesirably low results, those in sector VI also were produced in a rather slow and inefficient manner.



(a) Scatter plot: Athos in FbAf20

		Time			
		< 30	[30, 60)	≥ 60	Σ
Pts	≥ 75	0	3	24	27
	< 75	1	1	6	8
	Σ	1	4	30	35

(b) Cross table: Athos in FbAf20



(c) Scatter plot: JSprit in FbJf20

		Time			
		< 30	[30, 60)	≥ 60	Σ
Pts	≥ 75	0	1	14	15
	< 75	5	4	14	23
	Σ	5	5	28	38

(d) Cross table: JSprit in FbJf20

Figure 7.6: Score achieved in relation to survey time for Athos and JSprit used as a first approach in Friedberg, 2020.

a first approach, there was only one participant who spent less than 30 minutes answering the Athos questions. This participant scored 35 points which is a slightly better result than those found in the JSprit group for participants that rushed their answers. It can be assumed that participants in this group did not put in their best effort to correctly solve the tasks and merely guessed the answers.

Both compared groups had nearly the same number of participants who spent between 30 and 60 minutes on the tasks. For Athos, there were 4 participants in that category, for JSprit 5 participants were recorded. An important difference, however, is in the number of points that these participants scored: With Athos, 3 participants achieved more than half of all points (92, 102, 116) and only one fell below this threshold (61). With JSprit, on the other hand, there was only one participant who

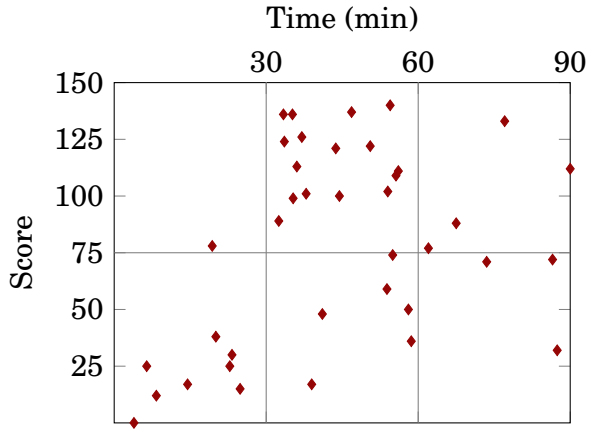
exactly scored 75 points whereas the other 4 participants fell below that mark (29, 30, 45, 48).

As concerns those participants that spent more than an hour on the respective language section, the numbers of both groups were also practically identical: in the Athos group 30 participants spent more than 60 minutes on the tasks and in the JSprit group 28 participants did the same for the JSprit questions. Again, the difference is in the score achieved by participants of both groups: With Athos, a marked 24 participants managed to score more than 75 points. With JSprit this number dropped to a mere 14 participants. With Athos, only 6 participants that spent more than 60 minutes on the tasks scored less than half the points. With JSprit, there were 14 participants that did not manage to score at least 50 per cent of the points in more than 60 minutes.

There is also a difference in the top results achieved with either approach: with Athos, the top five scores were 146, 136, 131, 130, and 129 points. With JSprit, the five top-scoring participants achieved 117, 116, 115, 113, and 103 points. In other words, even the highest score achieved with JSprit is 12 points less than the fifth best score obtained with Athos.

7.1.4.2 Results from Friedberg with second approach

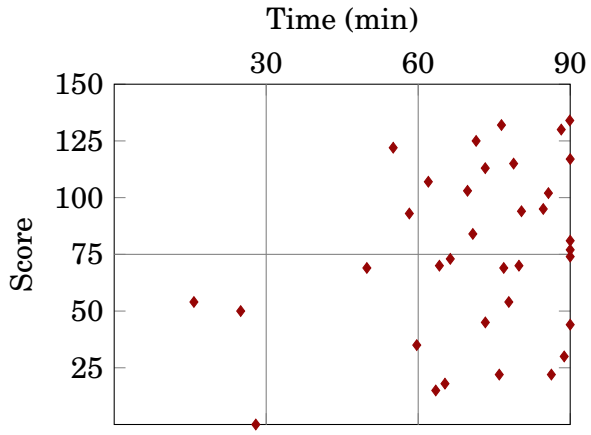
The results achieved by participants in Friedberg in 2020 with both Athos and JSprit as a second approach is shown in Figure 7.7. The overall numbers of participants that scored more than 75 points is rather similar: with Athos, 21 participants obtained more than 75 points whereas 17 participants managed to pass this threshold with JSprit. Though the number for Athos in this regards is slightly higher than the number for JSprit, it also has to be considered that there were more participants in the group that used Athos second. Hence, there is only a slight advantage for Athos in regard to passing participants. Consequently, the number of participants with less than 75 points is also somewhat similar with 17 participants failing the 75 points mark using Athos as the second approach and 18 participants short of this mark with JSprit as the second approach.



(a) Scatter plot: Athos as second in FbJf20

		Time			
		< 30	[30, 60)	≥ 60	Σ
Pts	≥ 75	1	16	4	21
	< 75	8	6	3	17
	Σ	9	22	7	38

(b) Cross table: Athos in FbJf20



(c) Scatter plot: JSprit second in FbAf20

		Time			
		< 30	[30, 60)	≥ 60	Σ
Pts	≥ 75	0	2	15	17
	< 75	3	2	13	18
	Σ	3	4	28	35

(d) Cross table: JSprit second in FbAf20

Figure 7.7: Score achieved in relation to survey time for Athos and JSprit used as a second approach in Friedberg, 2020.

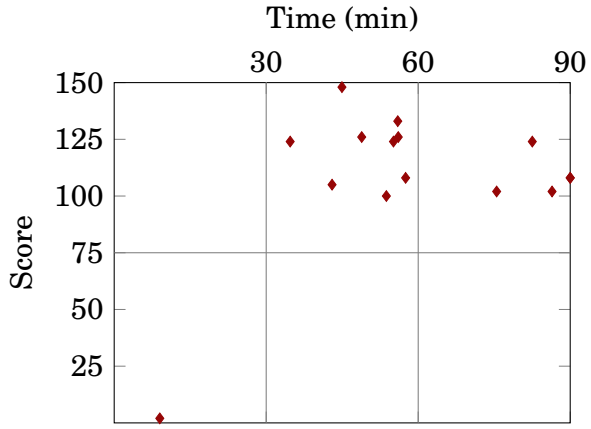
The obvious differences with both approaches is the time that participants spent for achieving their respective score. While nearly all participants with a positive result spent more than 60 minutes on the questions with JSprit, Athos users obtained their positive result considerably faster: 16 out of 21 participants that scored well were able to achieve their result in less than 60 minutes. With JSprit, only 2 out of 7 participants that finished in under 60 minutes passed the 75 points barrier and there was not a single participant who collected above or equal to 75 points in less than 30 minutes. With Athos, on the other hand, there was one participant that made the 50 per cent mark in under 30 minute, though only barely (78 points in 19 minutes and 21 seconds).

Though this participant was very efficient in obtaining the 75 points, the resulting 4.03 points per minute only suffice for the second place in the top five list of PPM achieved in this Athos second group (4.07, 4.03, 3.90, 3.87, 3.69). The top PPM value in the Athos group result from a participant who scored 136 points in 33 minutes and 24 seconds. The top five values with regard to PPM achieved with JSprit were (3.43, 2.22, 2.00, 1.75, 1.73). The best PPM score being produced by a participant who collected 54 points in just 15 minutes and 44 seconds. Though producing the score in an efficient manner, it still is a failing score and the participant barely achieved a third of all available points. The second best PPM score on this list was produced by a participant who scored 122 points in 55 minutes and 4 seconds.

What is striking to note is that for Athos there were as many as 8 participants who spent only little time on the test and achieved a result of less than 75 points. Compared to that only 3 participants who used JSprit as the second approach rushed to a result of less than 75 points. There are several possible reasons that may have led to this result. It is possible that these 8 participant disliked Athos and thus decided to not put too much effort in answering the questions. It may have also been the case that the shorter programs in Athos given them a false impression of security so that they answered the question with less focus and without double-checking their answers. It may also be the case that JSprit as a first approach was more exhausting than the other way round so that by the time participants took the second set of questions they did not feel motivated enough to spend a considerable amount of time in answering the questions correctly. Most likely, a mix of all these potential reasons is responsible for the observed outcome.

7.1.4.3 Results from Wetzlar with first approach

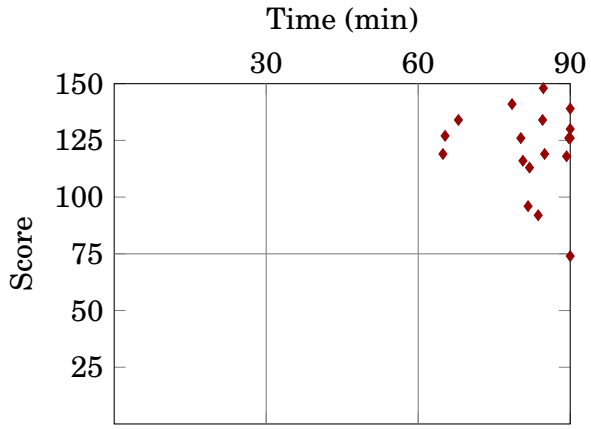
In Figure 7.8 the results for both approaches applied as a first approach in the Wetzlar 2020 study are depicted. Most obviously, in each group all but one participant passed the mark of 75 points. The failing participant of the Athos group merely



(a) Scatter plot: Athos as first in WzAf20

		Time			
		< 30	[30, 60)	≥ 60	Σ
Pts	≥ 75	0	9	5	14
	< 75	1	0	0	1
	Σ	1	9	5	15

(b) Cross table: Athos as first in WzAf20



(c) Scatter plot: JSprit first in WzJf20

		Time			
		< 30	[30, 60)	≥ 60	Σ
Pts	≥ 75	0	0	17	17
	< 75	0	0	1	1
	Σ	0	0	18	18

(d) Cross table: JSprit first in WzJf20

Figure 7.8: Score achieved in relation to survey time for Athos and JSprit used as a first approach in Wetzlar, 2020 (Note: two participants scored exactly 108 points in the full 90 minutes, thus the graphic shows only four diamonds in the respective sextant).

scored 2 points in exactly 9 minutes.[4] In the JSprit group, the failing participant was only one single point short of scoring half of all points. While this participant did not perform as well as their[5] peers, it was most likely not for lack of trying since the participant spent the maximum allowed time of 90 minutes on solving the tasks (see also Section 7.4).

The plotted data points also indicate that there was a noticeable difference in the time required by participants to obtain their scores. With JSprit, all 17 passing participants required more than 60 minutes to finish the questions. The fastest

[4] Inclusion of this participant results from the fact that the pre-defined inclusion criteria (see Section 6.7.4) were designed to allow for a broad population of participants that were only required to try to answer the questions and show a comparable effort in doing so for both approaches.

[5] The usage of the 'singular they' is intentional here.

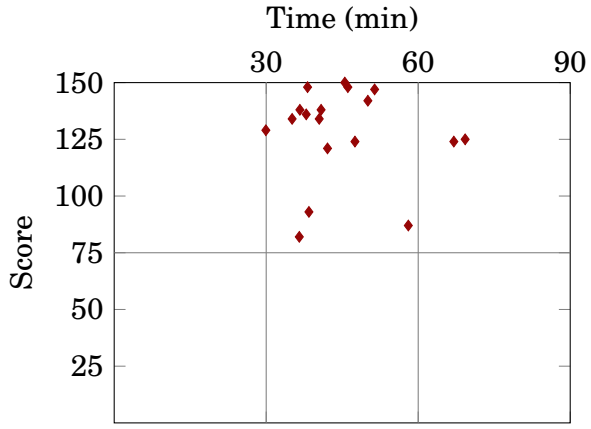
participant in the JSprit group spent around 65 minutes on the task awarding them 119 points. By contrast, with Athos there were 9 passing participants who completed the questions in less than 60 minutes. The fastest participant even finished the Athos question in 34 minutes and 44 seconds scoring as many as 124 points. A look at the top five values for PPM in both groups further indicates that Athos might have a positive effect on participants' efficiency. For Athos, 3.57, 3.29, 2.58, 2.44, and 2.38 were the top five PPM values. for JSprit the top five values in this regard were 1.97, 1.94, 1.83, 1.80, 1.75. With Athos, the most efficient participant succeeded in scoring an impressive 148 points in 44 minutes and 57 seconds. With JSprit, the highest PPM value was produced by a participant who also achieved an exceptional 148 points in 84 minutes and 42 seconds.

Though PPM seems to be in favour of Athos, the data also show that the best participants of each group achieved higher scores with JSprit. A brief look on the top five scores confirms this visual impression (148, 133, 126, 126, 124 for Athos and 148, 141, 139, 134, 134) for JSprit. In conclusion it can be said, that in these two groups in which the approaches were compared as first approaches, the overall scores obtained are in favour of JSprit. However, it can also be observed that Athos has the potential to increase participants efficiency by reducing the time required to finish semantically/structurally similar problems.

7.1.4.4 Results from Wetzlar with second approach

Figure 7.9 details the results participants from Wetzlar produced with both approaches applied second. As can be seen in the plots, participants from both groups did reasonably well. All but one participants succeeded in scoring at least 75 points. The failing participant is the same participant who also produced a very low score with Athos as their first approach, though the achieved score showed a slight improvement in comparisonto the result with Athos as a first approach[6]. In both groups

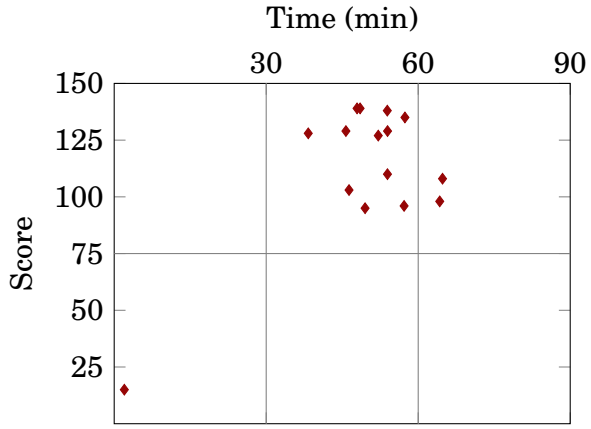
[6]From 2 points with Athos first to 15 points with JSprit second. However, given that the participant spent only 2 minutes on the tasks, it is likely that the participant merely guessed the answers of most tasks. Different in-/exclusion rules, would likely have eliminated this case from the set of results. With the defined rules, however, the case was to be included (see Section 7.4).



(a) Scatter plot: Athos as second in WzJf20

		Time			
		< 30	[30, 60)	≥ 60	Σ
Pts	≥ 75	1	15	2	18
	< 75	0	0	0	0
	Σ	1	15	2	18

(b) Cross table: Athos as second in WzJf20



(c) Scatter plot: JSprit second in WzAf20

		Time			
		< 30	[30, 60)	≥ 60	Σ
Pts	≥ 75	0	12	2	14
	< 75	1	0	0	1
	Σ	1	12	2	15

(d) Cross table: JSprit second in WzAf20

Figure 7.9: Score achieved in relation to survey time for Athos and JSprit used as a second approach in Wetzlar, 2020.

participants were able to score a substantial number of points within a relatively small amount of time. With Athos, 16 participants scored above 75 points in less than an hour, and one participant even managed to score 129 points in less than 30 minutes (29 minutes and 56 seconds). Though no participant from the JSprit group finished before the 30 minutes mark, the participants who used JSprit as a second approach still were rather efficient. 12 of 14 successful participants required less than 60 minutes to provide their answers. Two participants required more than an hour to finish, though both came quite close to the 60 minute mark (around 64 minutes each).

In the Athos first group one participant managed the feat of scoring a perfect 150 points. For this, the participant required 45 minutes and 31 seconds results in 3.3 PPM. Two more participants nearly also managed the perfect score with 148 points

each. One of these participants required 46 minutes and seven seconds for this score and the other participant even achieved the score in 38 minutes and 10 seconds. In the JSprit group, two participants tied for the best score at 139 points. Both participants also achieved this score in a virtually identical amount of time (47m 56s and 48m 32s). The third best-scoring participant in this group was awarded 138 points for which they spent 53 minutes and 54 seconds on the questions.

As regards the efficiency of participants, the top five participants from the Athos group scored 4.31, 3.88, 3.82, 3.77 and 3.59 PPM; for JSprit, the top five PPM achieved were 7.5, 3.34, 2.90, 2.86, and 2.82. The outlying 7.5 PPM result from the participant who spent only about two minutes on answering the questions because the participant managed to score 15 points in that time. Though this skews subsequent significance tests, the skewing occurs *against* Athos and would thus not pose a threat to the validity of a finding that showed an overall better efficiency for Athos (also see Section 7.4). Except for this special case, the observed top five PPM values are all in favour of Athos which indicates again that Athos might enhance users' efficiency.

The observed results also require a brief discussion on the different results when used as a first or second approach: Especially JSprit as a second approach, when compared to the group that applied JSprit as a first approach (see Figure 7.8), shows a markedly improved efficiency. For Athos as a second approach there was no such obvious acceleration in time required to solve the questions. However, for Athos as a second approach it appears that the achieved correctness increased compared to the group that applied Athos as a first approach. This can partly be explained with the language application skills of the respective group members. But it should also be considered an indicator for learning effects that must be considered when interpreting the results of within-subjects comparisons.

7.1.4.5 *Between subjects significance test on observed efficiency*

The between-subjects comparison of the observed efficiency measured in PPM was tested for statistical significance using the Mann-Whitney U test. Table 7.4 reports on the results obtained from these tests. In Friedberg, the group that used Athos first achieved significantly higher PPM with Athos (Mdn 1.22) than they did with JSprit (Mdn 0.87). Participants that used Athos second (Mdn 1.81) also obtained significantly higher PPM than with JSprit (Mdn 1.12). In Wetzlar, the group that used Athos first (Mdn 1.88) achieved a higher PPM than the JSprit first (Mdn 1.44) group, but the result was not statistically significant. Athos as a second approach in Wetzlar (Mdn 3.04) also resulted in a higher efficiency than with JSprit second (2.39) though not at a statistically significant level.

These results indicate that Athos is the approach that enables users to be more efficient at comprehending and creating VRPTW models. Out of four conducted between-subjects comparisons two showed with statistical significance that Athos users were able to achieve a higher value in PPM than participants that used the baseline approach.

7.1.4.6 *Witin subjects siginificance test on observed efficiency*

In addition to the between-subjects comparison presented in the previous paragraph, the obtained data were also used to conduct a within-subject comparison. Table 7.5 reports on the results obtained with Wilcoxon's signed rank test. Participants from

Table 7.4: Between subjects comparison of observed correctness using the Mann-Whitney U test.

	N		Mdn_{Score}		Mdn_{Rnk}				
	Athos	JSprit	Athos	JSprit	Athos	JSprit	U	Z	p
Fb as first	35	38	1.22	0.87	45.370	29.290	372.0	−3.235	**.001**
Fb as second	38	35	1.81	1.12	44.170	29.210	392.5	−3.009	**.003**
Wz as first	15	18	1.88	1.44	20.070	14.440	89.0	−1.663	.096
Wz as second	18	15	3.04	2.39	19.890	13.530	83.0	−1.880	.060

Friedberg who used Athos first (Mdn 1.22) produced significantly more PPM then with JSprit second (Mdn 1.12). When Athos was used second (Mdn 1.81) the scored PPM were also significantly higher than with JSprit first (Mdn 0.87). In Wetzlar, with JSprit as a second approach (Mdn 2.39) participants obtained significantly more PPM than with Athos (1.88). The other group that started out with JSprit (Mdn 1.44) scored significantly more points per minute when using Athos second (Mdn 3.04).

With one test significantly in favour of JSprit and three tests significantly favouring Athos, these results can safely be interpreted as evidence for Athos' potential to enhance users' efficiency. As was already discussed in Section 7.1.3.4, the results also indicate that an application library might still be a sensible approach for software developers that are experienced and satisfied with the underlying GPL. However, it is to be noted that with the second approach there might also be learning effects affecting the results for both approaches.

7.1.5 Results in terms of user satisfaction

This section presents the evaluation results in terms of user satisfaction (see Section 6.6.1.1). To gain insight into this aspect on the usability of the compared approaches, participants were presented five positive statements on different language characteristics. For both Athos and JSprit, participants were then asked to state their level of agreement to these statements applied to the respective approach.

Table 7.5: Within subjects comparison of the achieved PPM using Wilcoxon signed-rank test.

			Mdn_{Score}				
	n	Ties	Athos	JSprit	W	Z	p
Fb Athos first	35	0	1.22	1.12	152.00	-2.670^a	**.008**
Fb JSpirt first	38	0	1.81	0.87	41.00	-4.778^a	**.000**
Wz Athos first	15	0	1.88	2.39	21.00	-2.215^b	**.027**
Wz JSprit first	18	0	3.04	1.44	0.00	-3.724^a	**.000**

[a] Based on positive ranks

[b] Based on negative ranks

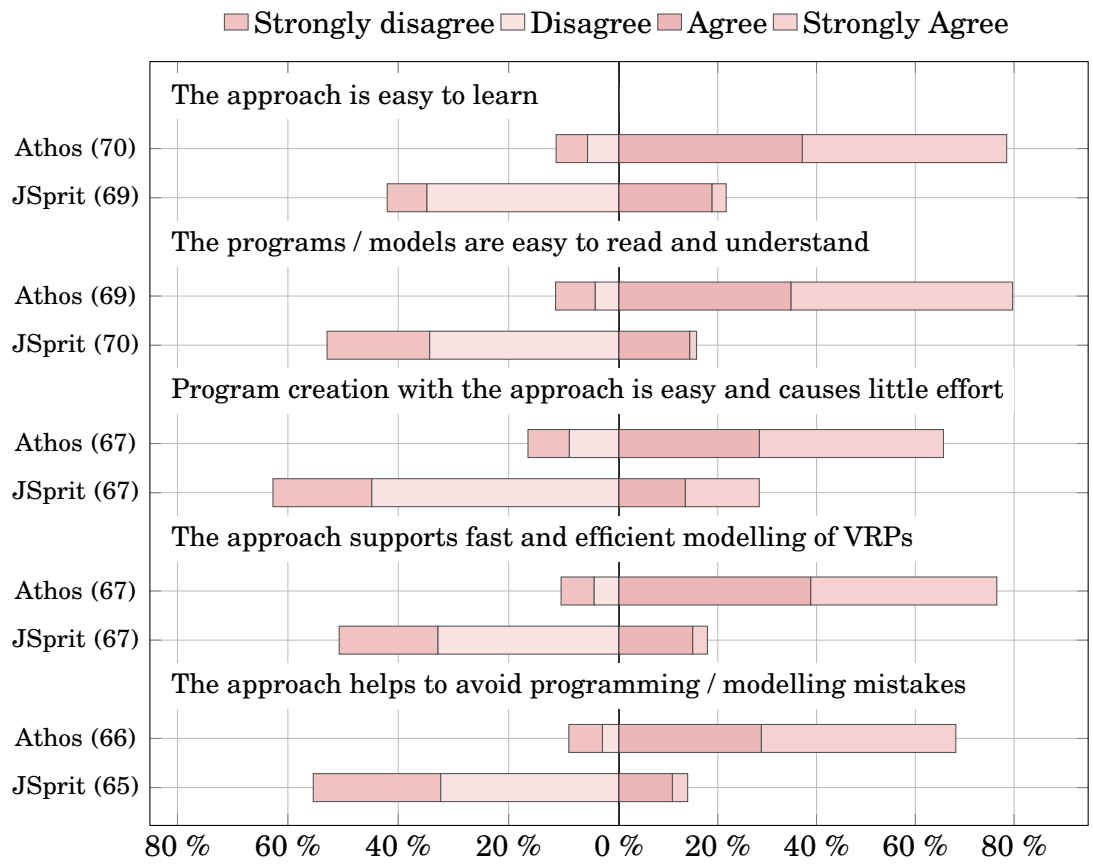


Figure 7.10: Overview on how participants from Frieberg in 2020 perceived working with Athos and JSprit by expressing their level of agreement to positive statements about both approaches.

The statements concerned the perceived learnability, the understandability, the convenience experienced in the creation of models, the ensuing speed of development and the support in the creation of correct models. Participants could express their degree of agreement by means of a five-point Likert scale. Section 7.1.5.1 presents the results obtained from the Friedberg study group, Section 7.1.5.2 then provides an overview on the results from the Wetzlar study.

7.1.5.1 User satisfaction in Friedberg

Figure 7.10 illustrates the level of agreement that participants from Friedberg provided on five positive statements on different language characteristics. The bars that extend to the left illustrate the proportions of participants who *disagreed* or *strongly disagreed* to the respective statement. The bars extending to the right rep-

resent *agreement* or *strong agreement*. The percentages were calculated based on the number of participants who provided a definitive statement so that opt-out answers were not considered. The percentage of participants who took a neutral stance toward the respective statements is thus the missing difference between accumulated displayed percentages to the full one hundred percent. For both approaches, the absolute number of participants who provided an answer to the respective statement is indicated in brackets on the left side of the illustration.

The statement that Athos was easy to learn was accepted by the vast majority of participants (≈ 82%). More than one-third of the provided answers indicated agreement (≈ 37.1%), and more than 40 % even expressed strong agreement to this statement on Athos. Only a small number of participants rejected this statement by expressing either disagreement (≈ 5.7%) or strong disagreement (≈ 5.7%). By contrast, with the same statement made on JSprit, not even one in four participants would agree. Around 42 % percent of all participants even would explicitly disagree (≈ 34.%) or even state strong disagreement (≈ 7.2 %).

The results on the other statements were very similar. Nearly 80% agreed to the statement that Athos models were easy to read and understand whereas for JSprit this statement was rejected by more than half of all participants (≈ 52.9 %). Athos had its weakest result when participants were asked whether they would agree that Athos models can be easily created. However, still nearly 2 out of 3 participants would support this claim (≈ 65.7% with either agreement or strong agreement). For JSprit, only around 15 % of all participants were willing to agree to this statement and nearly 2 out of 3 participants would deny that JSprit allows for easy VRP modelling (disagreement ≈ 44.8 %, strong disagreement ≈ 17.9 %).

The presented data undoubtedly show that Athos was the approach subjectively favoured by participants of the Friedberg study. The overwhelming majority recognised Athos as an approach that supported them in the modelling of VRPTWs through a moderate learning curve, conveniently readable and writeable models, and a syntax that made the introduction of modelling mistakes less likely.

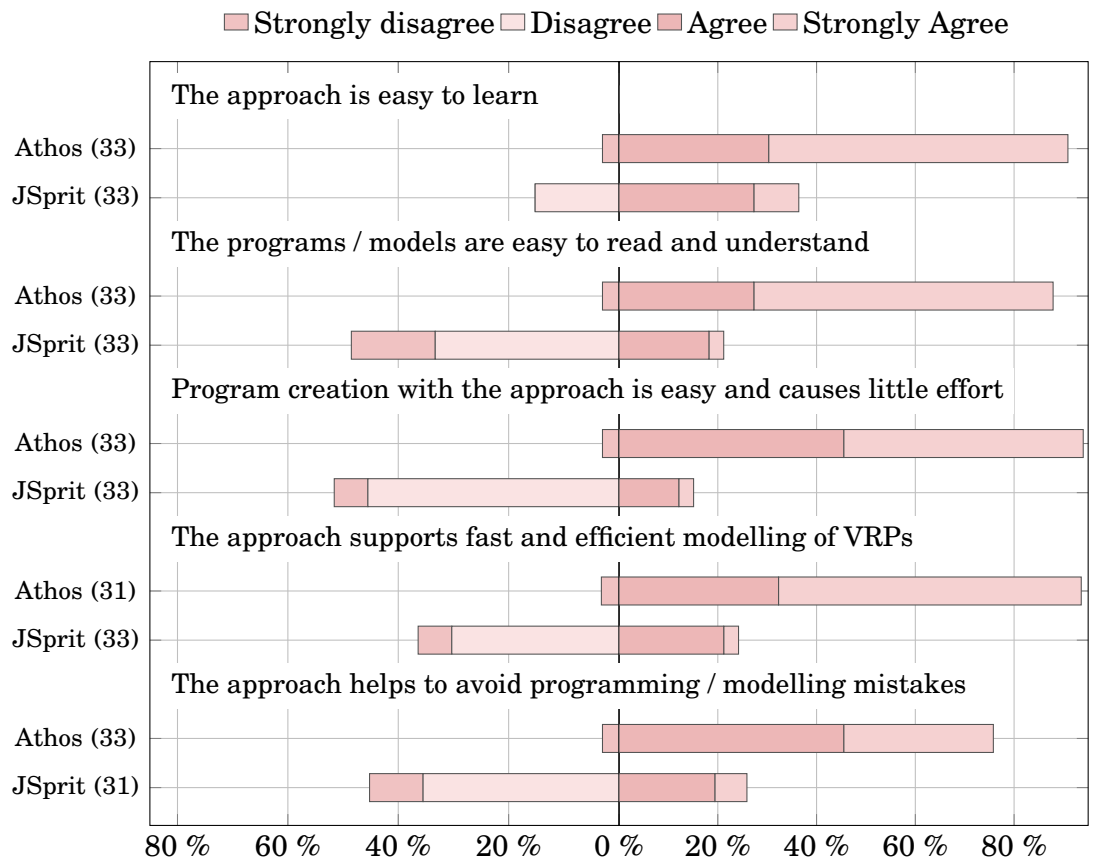


Figure 7.11: Overview on how participants from Wetzlar in 2020 perceived working with Athos and JSprit by expressing their level of agreement to positive statements about both approaches.

7.1.5.2 User satisfaction in Wetzlar

For participants in the Wetzlar study, Figure 7.11 summarises their level of agreement to the presented statements. As can be seen from the graphic, Athos undoubtedly met with the approval of these more experienced users (see Section 7.1.2). While the statement on easy and effortless program creation was the one users from the Friedberg study were most reluctant to agree to, in Wetzlar an overwhelming 94 % assented to that statement (agree ≈ 45.5%, strong agree ≈ 48.5%). The statement that Athos provides support for fast and efficient modelling of VRPs found similar strong approval among participants from Wetzlar: around 32.3 % of the participants agreed to it and over 60 % even expressed their strong agreement.

Statements on easy learnability and understandability resonated similarly well among Wetzlar participants. Around 90 % of all participants agreed or strongly

agreed that Athos is easy to learn and its models are easy to read (≈ 90.9%, and ≈ 87.9%, respectively). The lowest level of agreement was observed for the statement on Athos' support in the creation of correct models. Though the level of agreement to this statement is somewhat short of those observed for the other statements, 3 out of 4 participants would affirm this claim (agreement ≈ 45.5 %, strong agreement ≈ 30.3 %).

The results for JSprit, on the other hand, were considerably less supportive. The highest level of agreement was achieved with the statement on easy learnability. More than 1 in 3 participants perceived JSprit to be as rather simple to learn (agreement ≈ 27.3 %, strong agreement ≈ 9.1 %). The only other statement that found agreement among upward of 25% of all participants was the statement that JSprit helped users avoid modelling mistakes. Especially the statement that JSprit was easy and convenient to use was refused by a large number of participants (disagree ≈ 45.5 %, strong disagree ≈ 6.1 %).

The data obtained from the Wetzlar study also are strongly in favour of claims that Athos provides a high level of usability in terms of user satisfaction. The obtained results show that participants consider Athos to be the superior approach in all characteristics they were asked on. Though it is possible that there are additional language aspects that were not covered by the questions in the survey, the presented questions do cover the important characteristics of learnability, readability and applicability. Therefore, the data can be considered to provide strong evidence for a high level of user satisfaction brought about by the Athos approach.

7.2 Results of the replication study

7.2.1 Application of exclusion and inclusion criteria

Table 7.6 provides a report on the number of original cases as well as on the number of cases that were excluded by one of the filters applied. The meaning of the table columns was discussed in Section 7.1.1.

Table 7.6: Number of cases excluded from and included in the analysis of the 2021 study.

		Non-attempt filter						
Subgroup	Pre	#NA_AT	#NA_JS	OR*	AND	OL_TME	OL_SCR	Included
FbAf21	61	12	27	27	12	1	0	33
FbJf21	49	17	11	18	10	1´	0	30
WzAf21	20	5	6	6	5	1	1	12
WzJf21	19	4	2	4	2	0	0	15

Similar to the 2020 study, the Friedberg study group produced a larger absolute number of cases that hat to be excluded from the study. What was different in the 2021 study is that participants from Wetzlar also submitted a substantial number of cases in which more than seven questions remained unanswered. Especially in relation to the smaller number of participants from Wetzlar the comparatively high number of removals was somewhat surprising. A possible explanation might be that they were told by participants from the 2020 study that participation in the study was indeed not a mandatory requirement to pass the exam at the end of the semester.

7.2.2 Demographic data

The graphs shown in Figure 7.12 show a summary of the answers participants from the Friedberg 2021 study group gave on their prior programming knowledge. They are structurally similar to those graphs discussed for the 2020 study groups (see Section 7.1.2.2 and Section 7.1.2.1) so that they provide insight on the number of years of programming experience, the number of languages they had used until then, a self-assessment of their programming skills and a statement on their general interest in programming.

A comparison of participants' years as programmers shows that participants in the Athos group were moderately more experienced than their peers: 9 of the 31 (≈ 29%) participants had been programming for more than two years; in the JSprit group only 3 out of 29 participants had been programming for this long. However,

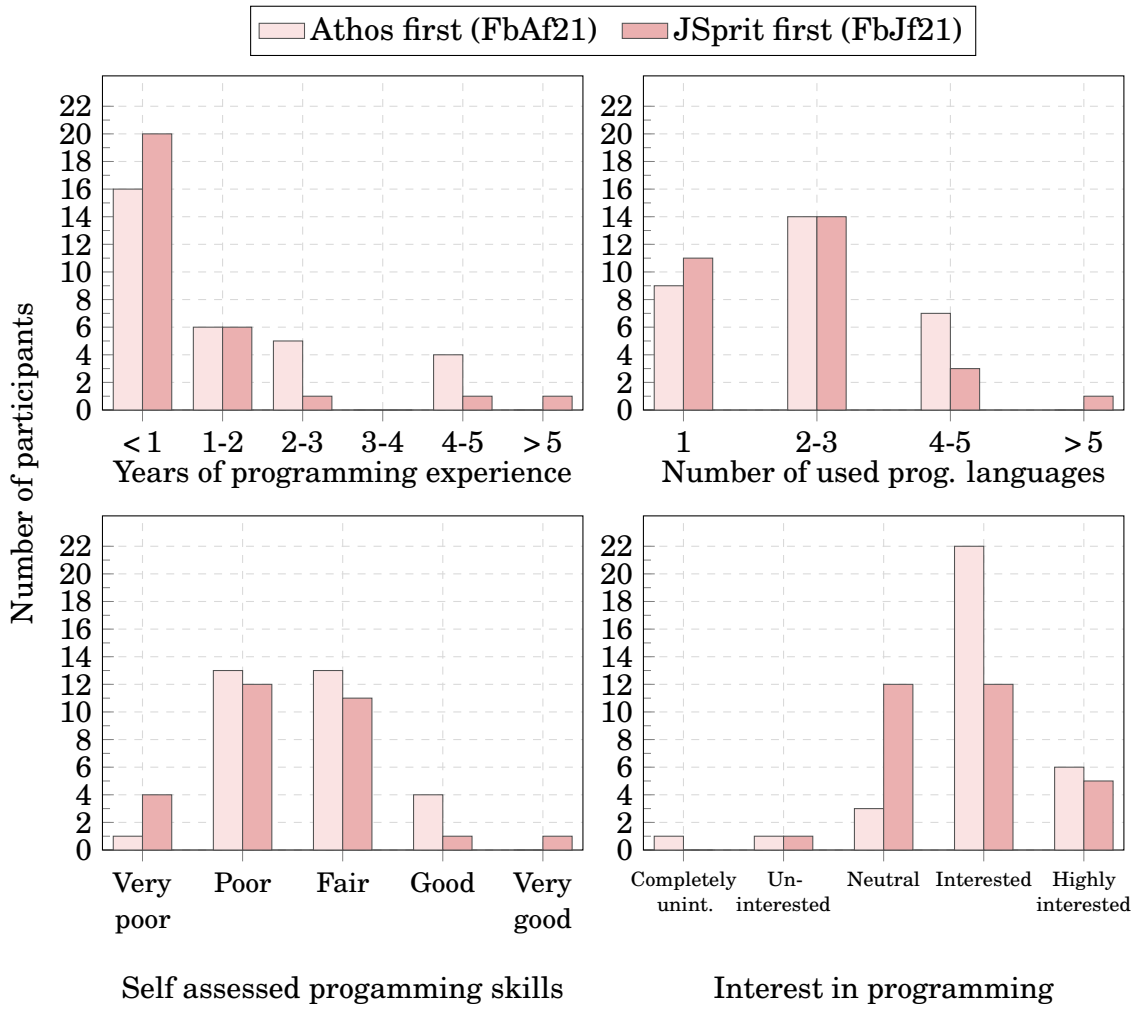


Figure 7.12: Comparison of participants' prior programming experience, number of programming languages used, self-assessed programming skills and general programming interest between the Friedberg 2021 study group using Athos first (▭) and the Friedberg 2021 study group using JSprit first (▭).

on a broad scale, participants from both groups were still rather inexperienced so that both groups should still be regarded to be comparable. Participants from both groups had contact with a similar number of computer languages. In both groups, most participants had not used more than three programming languages (Athos ≈ 77 %, JSprit ≈ 86 %).

The bar chart on the lower left of the figure shows that members from both groups had similar confidence in their programming skills though participants from the Athos group appeared to have a slight advantage in this category. In the Athos group, around 29 % of all participants doubted their skills. With around 55%, the number of participants that rated themselves 'poor' or even 'very poor'

programmers was moderately higher. The largest difference between both groups could be observed in terms of programming interest. In the Athos group, around 85 % of all participants claimed to be interested or highly interested in programming. Among the participants of the JSprit group this number was considerably lower as there only around 57% of all participants stated their interest in the topic.

In general, it can be said that based on the answers to the questions on participants' programming background those participants that used Athos as their first approach might have had a slight to moderate advantage. However, given the mostly subjective nature of these answers the observed differences here should not render a comparison of both groups skewed or pointless.

Figure 7.13 summarises the answers on prior knowledge for the Wetzlar study group. From all study groups, the Wetzlar 2021 study group shows the greatest imbalance in terms of years of prior programming experience. In the Athos first group, only around 17 % of the participants had less than two years of programming experience. By contrast, around 43 % of all participants from the JSprit first group had been programming for less than two years. This is a considerable difference that is likely to have an impact on the when discussing the results of the between-subjects comparisons.

Though the Athos first group consisted of more experienced programmers, participants in both groups had worked with a relatively large number of different computer languages. In the Athos group, 83 % of all participants had used four or more different languages. In the JSprit group, even nearly all participants had used at least four different languages (≈92.9 %). It has to be noted, though, that the Athos first group had a larger number of participants who had applied upwards of five different computer languages.

As concerns self-assessments in terms of skills and interest, results obtained from both groups were virtually identical. In both groups, the vast majority of participants considered themselves to be 'fair' or even 'good' programmers (Athos 91.7 %, JSprit 92.9 %). As for general interest in programming, nearly 92 % of all participant in the Athos group stated to be'interested' or even 'highly interested'

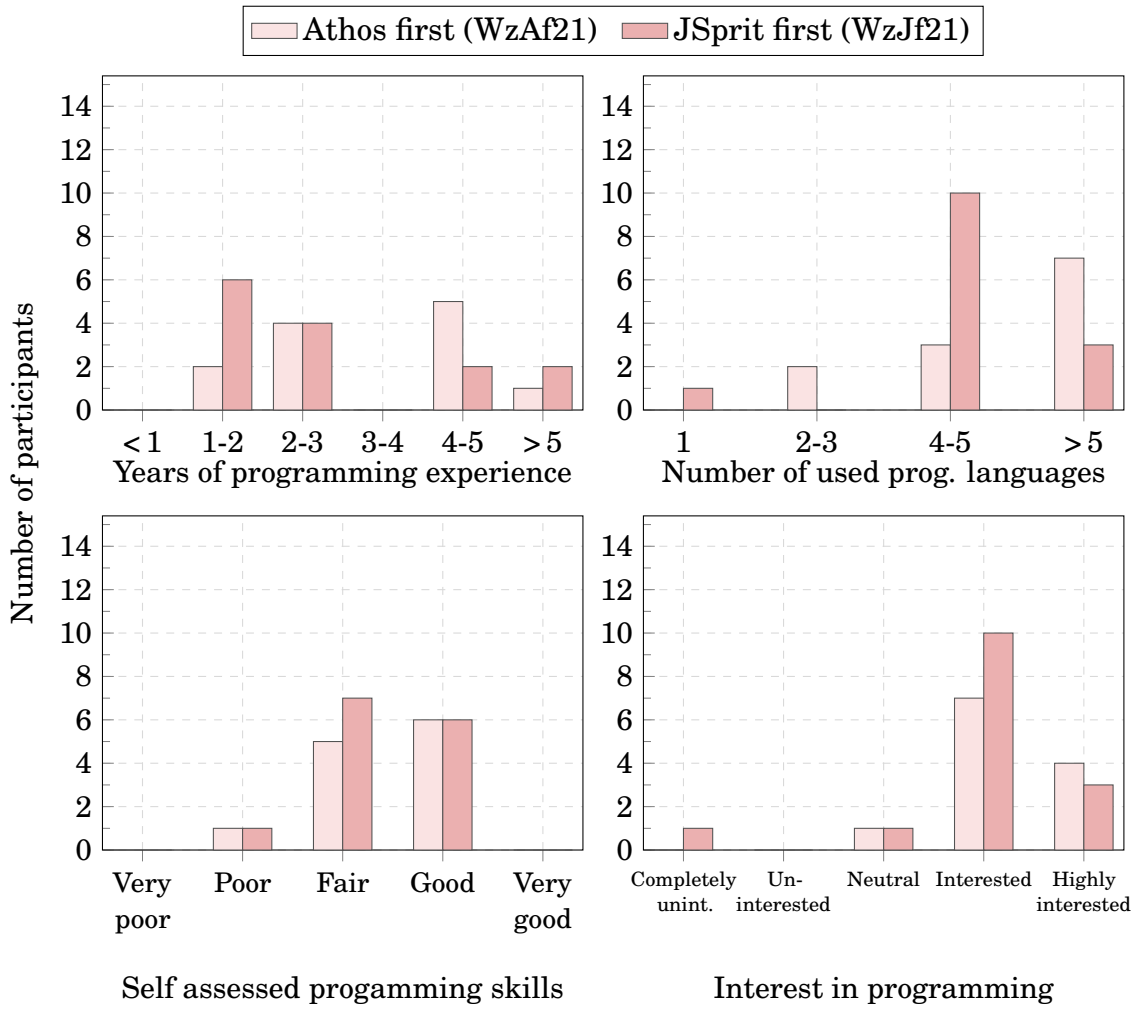


Figure 7.13: Comparison of participants' prior programming experience, number of programming languages used, self-assessed programming skills and general programming interest between the Wetzlar 2021 study group using Athos first ( ) and the Wetzlar 2021 study group using JSprit first ( ).

in programming. In the JSprit group, around 87 % of all participants made these claims.

The difference in years of experience warrants some consideration when comparing the results of both groups in terms of correctness and efficiency. It cannot be ruled out that the higher number of years that participants from the Athos group had been writing programs might affect the obtained results in some way. It is also to be noted that participants from Wetzlar were again more interested, experienced, and confident in the application of programming languages than participants from Friedberg. However, in the 2021 studies the gap in the categories 'years of experience' and 'interest' became smaller in comparison to the gap observed between the

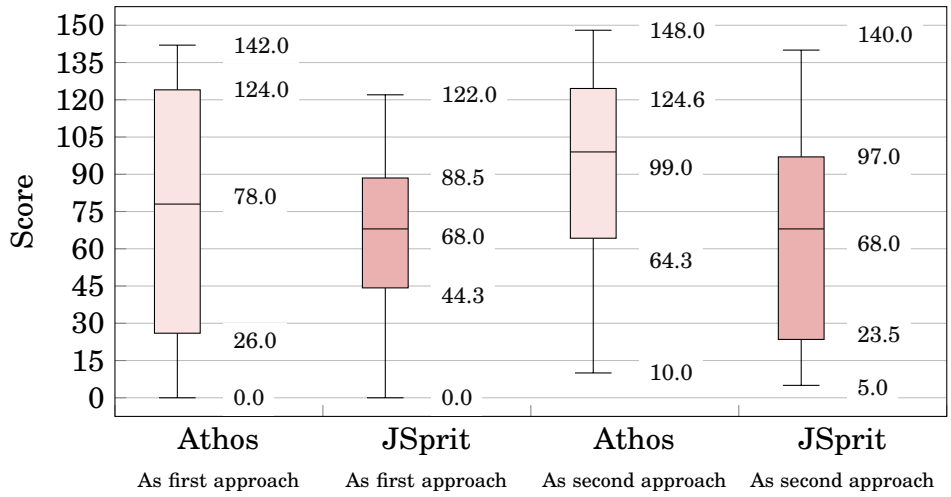


Figure 7.14: Boxplot showing the distribution of the scores achieved by participants of the study conducted in Friedberg in 2021 ($n_{\text{FbAf21}} = 33, n_{\text{FbJf21}} = 30$).

2020 study groups. This somewhat waters down the view in which participants from Friedberg represent domain experts and participants from Wetzlar represent experienced software developers. Though the prior knowledge in both study groups was not as markedly different as in 2020, they were still participants with more experience and enthusiasm towards the application of programming languages.

7.2.3 Results in terms of correctness

7.2.3.1 Results from Friedberg

Figure 7.14 shows the scores obtained in the subgroups of the Friedberg study group. Starting with a comparison of the results for both Athos and JSprit as first approaches, it is to be noted that the median of the Athos group was 10 points above the median of the JSprit first group. It can also be observed that the interquartile range with Athos was substantially wider than the one observed among participants of the JSprit group. This indicates that with Athos several users were very proficient whereas several others did not do well with the language. With Athos, the maximum score was 142 points compared to a maximum score of 122 points achieved with JSprit as a first approach. Though the wide dispersion among Athos users warrants further investigation (see Section 7.2.4), these results still are in favour of Athos as a first approach.

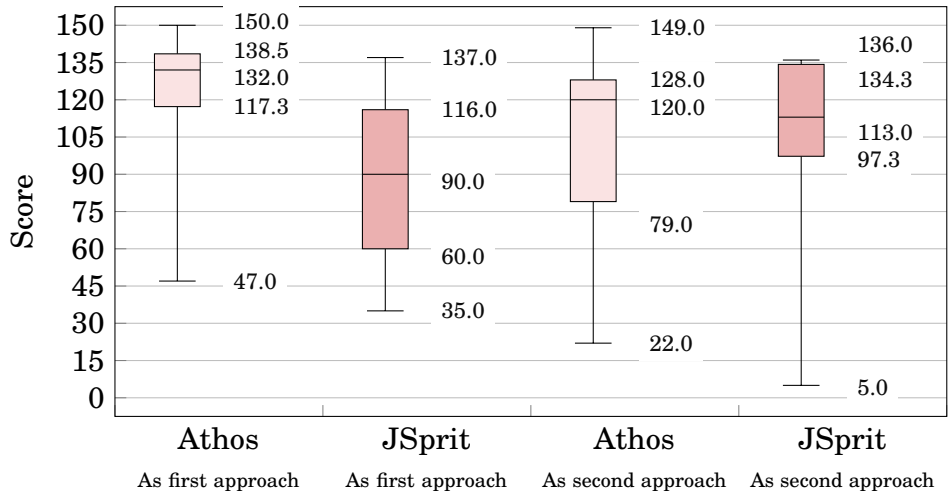


Figure 7.15: Boxplot showing the distribution of the scores achieved by participants of the study conducted in Wetzlar in 2021 ($n_{\text{WzAf21}} = 12, n_{\text{WzJf21}} = 15$).

A comparison of both approaches used second even more clearly points to Athos as the preferable approach. With Athos a median score of 99 points was achieved which is 31 points above the median achieved with JSprit. The maximum score with Athos was 148 points which is very close to the perfect 150. With JSprit the best score was 140 points and thus 8 points below the top score in the Athos group.

With Athos first and JSprit second the average score dropped by 10 points from 78 points to 68 points and the best score was reduced by 2 points. With Athos first the upper quartile was at 124 points and with JSprit second it was at 97 points. On the other hand, when JSprit was the first and Athos the second approach, the median increased by a marked 31 points from 68 to 99. With Athos second, the maximum score also increased by as many as 26 points from 122 to 148.

These data indicate that Athos enabled participants to increase the correctness of their answers. Both, the between subjects and the within subjects comparisons hint at Athos being the the superior approach in terms of correctness. In all discussed comparisons the average scores obtained with Athos showed to be superior to those obtained with JSprit.

7.2.3.2 *Results from Wetzlar*

Figure 7.15 displays the scores participants from the Wetzlar study group achieved with Athos and JSprit. When participants used Athos as the first approach, the median was at 132 points which is a considerable 42 points above the median of the group that used JSprit first. The median of the Athos group was only 5 points below the maximum score found in the JSprit group. The highest-score in the JSprit group was 137 points. In the Athos group the highest score was a perfect 150 points. From the box-and-whisker plots for both approaches as a first approach it is evident that the data are in favour of Athos as the first approach.

As a second approach, the median for Athos was also above the median observed for JSprit. Here, the difference between the medians of the two groups was 7 points. The best score achieved with a second approach was produced in the Athos group with a near-perfect 149 points.

The usage of Athos as a first and JSprit as a second approach led to a decline in the average and maximum scores achieved. The median dropped by a marked 19 points and the highest score reduced from 150 points to 136 points. When JSprit was used as the first and Athos as the second approach, the median rose sharply from 90 to 120 points and the best observed score also increased from 137 to 149 points.

The data observed in the Wetzlar subgroup are clearly in favour of Athos. These data strongly hint at Athos being the approach that enables users to produce substantially better results in terms of correctness. Either the within subjects and the between subjects comparisons in this group leave little doubt that Athos was the approach to be preferred in order to produce markedly better results.

7.2.3.3 *Between subjects significance tests on observed correctness*

In analogy to the original study (see Section 7.1.3.3), significance tests on the between-subjects data were conducted the results of which are summarised in Table 7.7. With Athos first, participants from Friedberg on average scored higher (Mdn 78) than they did with JSprit (Mdn 68). This result, however was not statistic-

Table 7.7: Between subjects comparison of the achieved scores in the 2021 studies using the Mann-Whitney U test.

	N		Mdn_{Score}		Mdn_{Rnk}				
	Athos	JSprit	Athos	JSprit	Athos	JSprit	U	Z	p
Fb as first	33	30	78	68	33.920	29.880	431.5	−0.874	.382
Fb as second	30	33	99	68	36.900	27.550	348.0	−2.024	**.043**
Wz as first	12	15	132	90	19.040	9.970	29.5	−2.955	**.003**
Wz as second	15	12	120	113	13.630	14.460	84.5	−0.269	.788

ally significant. Used as a second approach, the scores obtained with Athos (Mdn 99) were significantly higher than the scores produced with JSprit (Mdn 68).

Observations from Wetzlar show a significantly higher score for the group that used Athos as a first approach (Mdn 132) than the group that used JSprit first (Mdn 90). Applied as a second approach, the scores for Athos (Mdn 120) were also higher than those for JSprit (Mdn 113) though not at a statistically significant level.

Just as in the original study, these data show that Athos can enhance the correctness of modellers in the specification and comprehension of VRPTW. Two out of four tests showed a significant advantage for Athos in this regard. Two additional comparisons were also in favour of Athos, though not statistically significant.

7.2.3.4 Within subjects significance tests on observed correctness.

Table 7.8: Within subjects comparison of the scores achieved in the 2021 studies using the Wilcoxon signed-rank test.

			Mdn_{Score}				
	n	Ties	Athos	JSprit	W	Z	p
Fb Athos first	33	0	78.0	68.0	138.5	−2.538[a]	**.011**
Fb JSpirt first	30	1	99.0	68.0	39.0	−3.861[a]	**.000**
Wz Athos first	12	0	132.0	113.0	8.0	−2.434[a]	**.015**
Wz JSprit first	15	0	120.0	90.0	13.0	−2.670[a]	**.008**

[a] Based on positive ranks

[b] Based on negative ranks

The results from the statistical tests that used the obtained data on achieved scores for a within-subjects comparison are presented in Table 7.8. In Friedberg, the group that used Athos first (Mdn 78) achieved significantly higher scores with this approach than they did when they used JSprit as their second approach (Mdn 68). The other group that began with the JSprit approach (Mdn 68) also achieved significantly higher scores with Athos (Mdn 99).

The data from Wetzlar present a similar picture. Using Athos first (Mdn 132) participants scored significantly higher than with JSprit second (Mdn 68). The other subgroup that used the approaches in the opposite order also achieved siginificantly higher scores with Athos (Mdn 120) than with JSprit (Mdn 90).

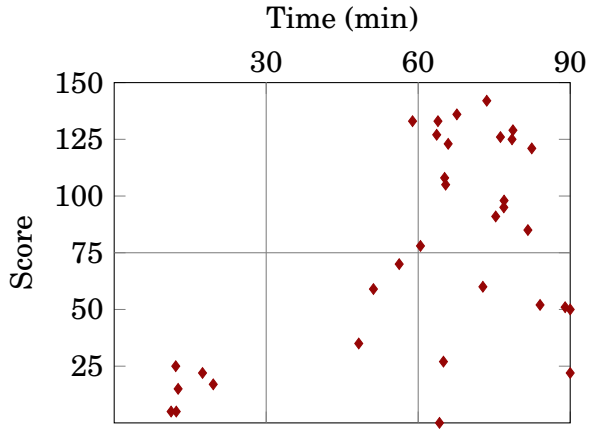
These data provide conclusive evidence that Athos aids participants in correct model comprehension and creation. With all four conducted tests showing significant advantages for Athos users, a strong claim for its application in the domain of VRPTW related problems can be made.

7.2.4 Results in terms of efficiency

7.2.4.1 Results from Friedberg with first approach

Figure 7.16 reports the results for both Athos and JSprit used as a first approach in the study conducted in Friedberg in 2021. Both plots show a rather broad dispersion of the data points. A first notable fact the high number of data points for Athos in sector III. There were 6 participants who scored relatively low and spent less than 30 minutes on answering the questions. The JSprit plot, by contrast, shows only three plot points in Sector III. Another striking result in the Athos group was produced by a participant who spend 64 minutes on the questions without scoring a single point.

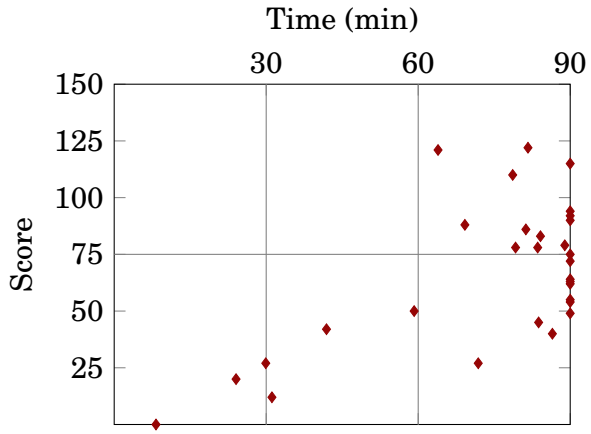
Both approaches had a comparable number of participants who spent more than 60 minutes on the questions but managed to obtain more than 75 points. The plot for Athos has 16 data points in Sector III and the plot for JSprit exhibits 14 data points



(a) Scatter plot: Athos as first in FbAf21

		Time			
		< 30	[30,60)	≥ 60	Σ
Pts	≥ 75	0	1	16	17
	< 75	6	3	7	16
	Σ	6	4	23	33

(b) Cross table: Athos as first in FbAf21



(c) Scatter plot: JSprit first in FbJf21

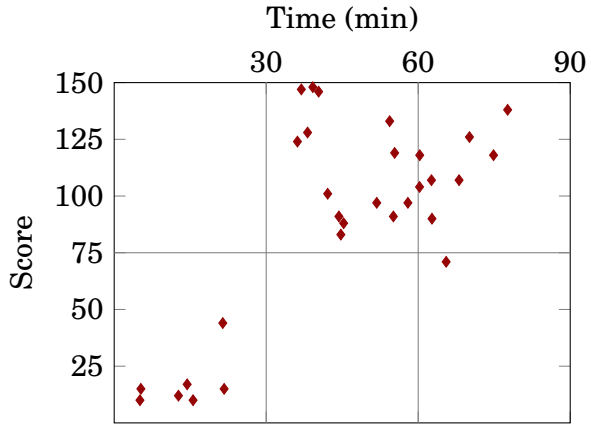
		Time			
		< 30	[30,60)	≥ 60	Σ
Pts	≥ 75	0	0	14	14
	< 75	3	3	10	16
	Σ	3	3	24	30

(d) Cross table: JSprit first in FbJf21

Figure 7.16: Scores achieved in relation to survey time for Athos and JSprit used as a first approach in Friedberg, 2021.

in that sector. Another remarkable observation is the high number of participants from the JSprit group that used the entire 90 minutes on answering the JSprit questions. 12 Participants from the JSprit group did not leave the question section on their own but had to be stopped by the maximum allowed time. With Athos, only 2 participants went the full temporal distance.

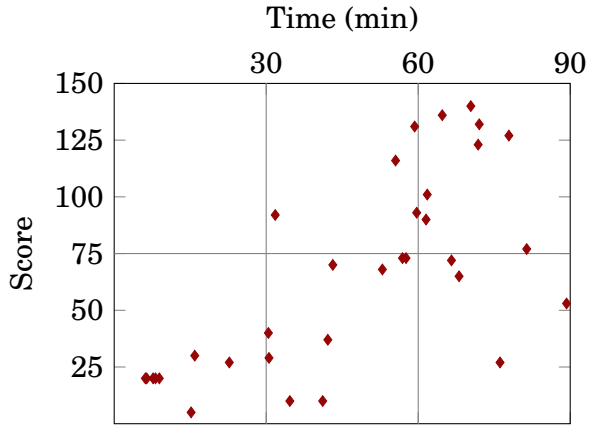
From the plot it can also be seen that top-scores for Athos were moderately higher than those of the JSprit group. The top five participants from the Athos group achieved 142, 136, 133, 133, and 129 points (with 73m 32s, 67m 38s, 58m 53s, 63m 53s, and 78m 42s being the respective times). The five highest scores observed in the JSprit group were 122, 121, 115, 110, and 94 (81m 40, 63m 55s, 90m, 78m 39s, 90m). This result again is in favour of Athos as it allowed for the highest scores



(a) Scatter plot: Athos as second in FbJf21

		Time			
		< 30	[30,60)	≥ 60	Σ
Pts	≥ 75	0	14	8	22
	< 75	7	0	1	8
	Σ	7	14	9	30

(b) Cross table: Athos as second in FbJf21



(c) Scatter plot: JSprit second in FbAf21

		Time			
		< 30	[30,60)	≥ 60	Σ
Pts	≥ 75	0	4	8	12
	< 75	8	9	4	21
	Σ	8	13	12	33

(d) Cross table: JSprit second in FbAf21

Figure 7.17: Scores achieved in relation to survey time for Athos and JSprit used as a second approach in Friedberg, 2021.

in the two compared groups. As concerns the efficiency of both approaches, the top 5 PPM scores for Athos were 2.26, 2.08, 2.06, 2.01, and 2.00. The top five results for JSprit were 1.89, 1.49, 1.40, 1.28 and 1.27. For Athos, the top PPM result came from a participant who scored 133 points in 58 minutes and 53 seconds. For JSprit, a participant who scored 121 points in 63 minutes and 55 seconds contributed the top PPM result. Thus, in conclusion it can be said that Athos results were superior in terms of the top scores in correctness and efficiency, but on the downside, Athos had twice the number of participants who scored very low in a short amount of time.

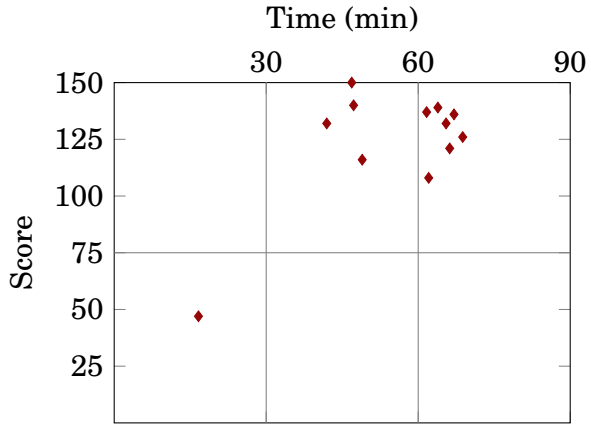
7.2.4.2 Results from Friedberg with second approach

Figure 7.17 provides an overview on the results obtained from the Friedberg replication study when Athos and JSprit were used as second approaches. On a general note it can be observed that the plot points for JSprit are substantially more dispersed than those in the Athos plot. A striking difference between the two plots is the fact that the JSprit plot shows 9 plot points in Sector IV whereas in the Athos plot this sector is completely empty. Similarly, Sector VI in the JSprit plot shows 4 data points whereas for the Athos plot there is only one points in this sector that barely missed the border to Sector III.

For both approaches, there is a similar number of participants whose did not score at least half of all points but who also spent less than 30 minutes on answering the questions. These data suggest that the usage of Athos might enhances users' correctness to a satisfactory level provided that they are willing to spend a certain minimum effort into the application of the language. This conjecture is also backed by the fact that with Athos upwards of two in three participants achieved at least 75 points while with JSprit the number of successful participants dropped to 36 per cent.

As regards the efficiency observed in this group, with Athos there were three participants that nearly achieved a perfect score in less than 60 minutes. The most efficient of these three participants scored 148 points in only 39 minutes and 12 seconds resulting in 3.78 PPM. For JSprit, the top scores in terms of PPM result from participants whose results are located in Sector IV of the plot (4 of the top five PPM results are located in Sector IV). The only top five PPM value not failing the 75 points mark was a participant who scored 92 points in 31 minutes and 47 seconds.

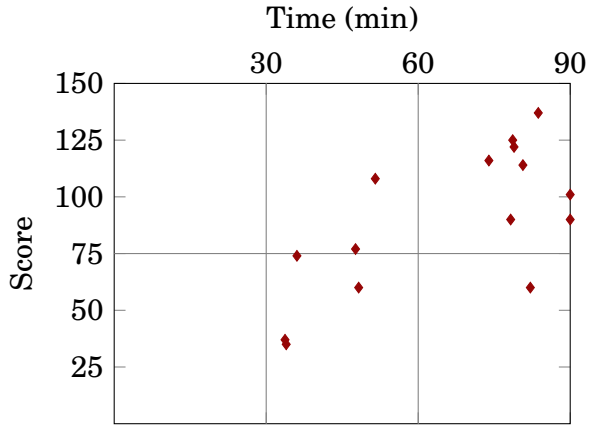
In conclusion it can be said that the data observed in this study group suggest that Athos has the potential to positively affect users' effectiveness and efficiency provided that they are willing to exert themselves to produce correct results. Not only does it have the potential to support average users to achieve at least satisfactory results, it might even boost the efficiency of already productive users.



(a) Scatter plot: Athos as first in WzAf21

		Time			
		< 30	[30, 60)	≥ 60	Σ
Pts	≥ 75	0	4	7	11
	< 75	1	0	0	1
	Σ	1	4	7	12

(b) Cross table: Athos as first in WzAf21



(c) Scatter plot: JSprit first in WzJf21

		Time			
		< 30	[30, 60)	≥ 60	Σ
Pts	≥ 75	0	2	8	10
	< 75	0	4	1	5
	Σ	0	6	9	15

(d) Cross table: JSprit first in WzJf21

Figure 7.18: Scores achieved in relation to survey time for Athos and JSprit used as a first approach in Wetzlar, 2021.

7.2.4.3 Results from Wetzlar with first approach

Figure 7.18 summarises the results for both approaches used as a first approach in the Wetzlar 2021 study group. Generally, the plot for JSprit shows a wider spread among the data points in comparison to the Athos plot. This results partly from a marked difference in the scores of those participants who spent 30 to 60 minutes on the respective questions. With JSprit, 4 out of 6 of these participants failed to score more than 75 points. While one of these participants only barely missed the 75 points mark by scoring 74 points, there was another participant who only just passed this mark by scoring 77 points. By contrast, with Athos, every participant who spent at least 30 minutes on the questions passed the 50 per cent threshold by a comfortable margin.

In terms of efficiency, with Athos the most efficient participant was also the most effective with a perfect score of 150 points in 46 minutes and 53 seconds thus scoring 3.2 PPM. The most efficient participant from the JSprit group scored 2.1 PPM by achieving 108 points in 51 minutes and 32 seconds. For JSprit, the highest scoring participant with 137 points in 83 minutes and 42 seconds ranks third in terms of efficiency with 1.64 PPM.

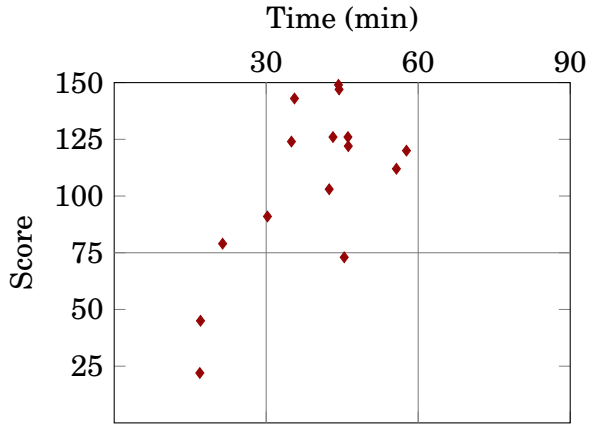
Generally, the data from this group also support the claim that Athos allows to effectively and efficiently address problems related to the modelling of VRPTW. The data also indicate that even though a certain minimum effort is required with the language, this effort is rewarded with satisfactory and even formidable results.

7.2.4.4 Results from Wetzlar with second approach

Figure 7.19 shows a summary of the results for Athos and JSprit as a second approach from the Wetzlar 2021 study group. Overall, it can be said that participants from both groups produced acceptable results. With JSprit, only one participant failed to score at least 75 points. With Athos, there were 3 participants who did not pass the 50 per cent threshold. Hence, it must be concluded that the number of failing participants of the compared groups is slightly in favour of JSprit. However, it is also to be noted that 2 of the 3 failing participants with Athos spent less than 30 minutes on the tasks (16m 54s and 17m 2s) and the third participant came very close to the 50 per cent mark with a score of 73 points.

In terms of successful participants, those participants who used Athos as the second approach appeared to have a slight advantage in terms of time required to finish the questions. With JSprit, three participants required more than 60 minutes to complete the questions, though 2 of these participants were very close to finishing within an hour (61m 3s and 62m 18s). By contrast, all participants from the Athos group managed to submit their answers before the 60 minutes mark.

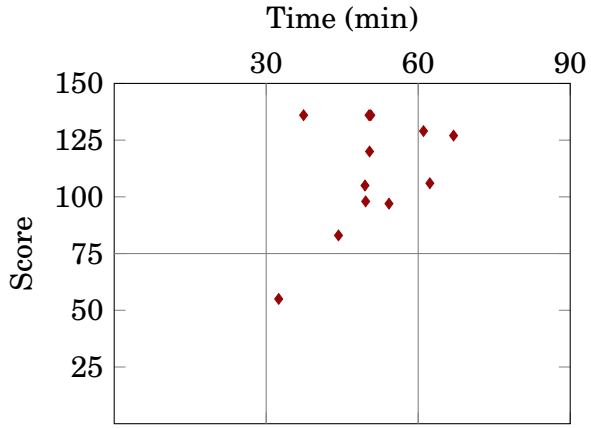
In terms of efficiency the most efficient participants from the respective group had similar results: the most efficient participant from the Athos group scored 143 points



(a) Scatter plot: Athos as second in WzJf21

		Time			
		< 30	$[30, 60)$	≥ 60	Σ
Pts	≥ 75	1	11	0	12
	< 75	2	1	0	3
	Σ	3	12	12	15

(b) Cross table: Athos as second in WzJf21



(c) Scatter plot: JSprit second in WzAf21

		Time			
		< 30	$[30, 60)$	≥ 60	Σ
Pts	≥ 75	0	8	3	11
	< 75	0	1	0	1
	Σ	0	9	3	12

(d) Cross table: JSprit second in WzAf21

Figure 7.19: Scores achieved in relation to survey time for Athos and JSprit used as a second approach in Wetzlar, 2021.

in 35 minutes and 35 seconds which equals 4.02 PPM. The most efficient participant from the JSprit group scored a close 3.64 PPM with 136 points in 37 minutes and 23 seconds. In the Athos group there were 2 more participants who scored an exceptional 149 points (44m 16s) and 147 points (44m 23s), which equals 3.37 and 3.31 PPM, respectively. In the JSprit group, there were two more participants who also scored 136 points one requiring 50 minutes and 18 seconds and the other 50 minutes and 39 seconds which amounts to 2.7 and 2.69 PPM, respectively.

From the data observed in these two groups it can be concluded that the results were not as distinct as those observed in the other groups. It must be stated that users in the JSprit group produced results that indicate an overall efficiency which is similar to the efficiency observed within the Athos group. However, there still

appears to be a slight advantage for users that apply Athos to problems related to the modelling of VRPTW, though it might be less distinctive.

7.2.4.5 *Between subjects significance tests on observed efficiency*

Table 7.9: Between subjects comparison of the scores achieved in the 2021 studies using the Mann-Whitney U test.

	N		Mdn_{Score}		Mdn_{Rnk}				
	Athos	JSprit	Athos	JSprit	Athos	JSprit	U	Z	p
Fb as first	33	30	1.24	0.87	37.500	25.950	313.5	−2.498	**.012**
Fb as second	30	33	1.86	1.56	37.070	27.390	343.0	−2.092	**.036**
Wz as first	12	15	2.20	1.41	20.670	8.670	10.0	−3.904	**.000**
Wz as second	15	12	2.73	2.04	16.600	10.750	51.0	−1.903	.057

The results of the between-subjects statistical tests in terms of observed efficiency measured in PPM are reported in Table 7.9. A comparison of both approaches used first in the Friedberg study group reveals that participants that used Athos (Mdn 1.24) were significantly more efficient than those that used JSprit first (Mdn 0.87). With Athos as a second approach, participants were also significantly more efficient (Mdn 1.86) than their peers who applied JSprit as the second approach (Mdn 1.56).

In the Wetzlar study group, the subgroup that used Athos first (Mdn 2.2) were significantly more efficient than the participants from the other subgroup that applied JSprit as their first approach (Mdn 1.41). When the approaches were used second, the achieved efficiency with Athos (Mdn 2.73) was higher than the efficiency produced with JSprit (Mdn 2.04) yet barely missed a significance level of 0.05 (p-value was 0.057).

Three of four conducted test provided significant evidence for the claim that Athos has the potential to boost users' efficiency. A fourth test also hinted at an improved efficiency ensuing from the usage of Athos but was only very close to being statistically significant. The conclusion drawn from the presented data is that there are very good arguments for the application of Athos to achieve high efficiency in tasks related to the modelling of VRPTWs.

7.2.4.6 *Within subjects siginificance test on observed efficiency*

Table 7.10: Among subjects comparison of the achieved PPM in 2021 using the Wilcoxon signed-rank test.

			Mdn_{Score}				
	n	Ties	Athos	JSprit	W	Z	p
Fb Athos first	33	0	1.2	1.6	158.0	−2.189[b]	**.029**
Fb JSpirt first	30	0	1.9	0.9	2.0	−4.741[a]	**.000**
Wz Athos first	12	0	2.2	2.0	31.0	−0.628[a]	.530
Wz JSprit first	15	0	2.7	1.4	0.0	−3.409[a]	**.001**

[a] Based on positive ranks

[b] Based on negative ranks

The results of the statistical tests for the within-subjects comparison of the observed efficiency measured in PPM are reported in Table 7.10. In Friedberg, the group that started with Athos (Mdn 1.2) was significantly more efficient with JSprit as their second approach (Mdn 1.6). However, the parallel group that used the approaches in reverse order was more efficient with Athos (Mdn 1.9) as their second approach than they were with JSprit as their first (Mdn 0.9).

As to the Wetzlar study group, here the group that started with Athos (Mdn 2.2) was slightly more efficient with Athos than they were with JSprit as their second approach (Mdn 2.0). This result, however, was nowhere near being statistically significant. When Athos was used as a second approach (Mdn 2.7) after JSprit was applied first (Mdn 1.4), users however showed to be significantly more efficient.

At first sight, these results convey a somewhat mixed picture on which approach is to be favoured. Two tests were significantly in favour of Athos, whereas one was significantly in favour of JSprit. However, it is to be noted here that the observed Z-scores indicate a very strong result in the tests where Athos was significantly superior, whereas the observed Z-score for the test where JSprit was significantly superior is a little weaker. Given that two tests favoured Athos and only one was in favour of JSprit, the data as a whole are still to be regarded to support Athos as the recommended approach for efficient modelling of VRPTWs.

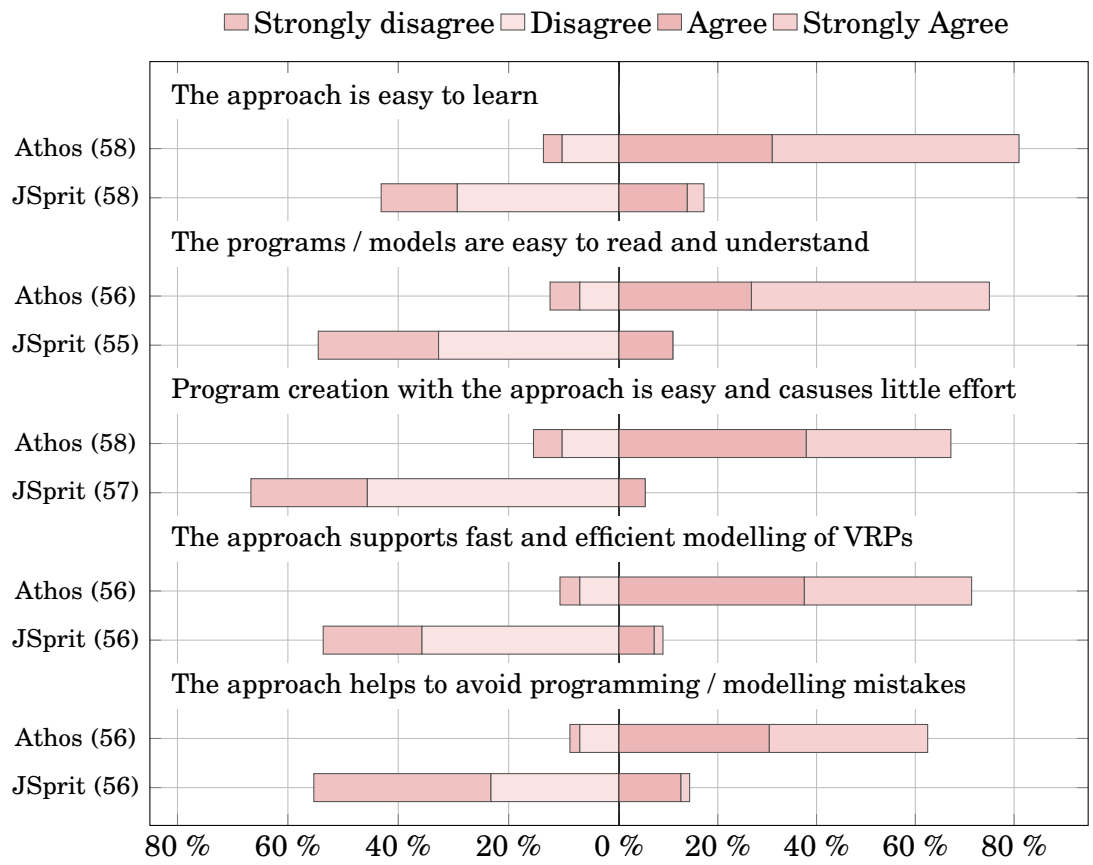


Figure 7.20: Overview on how participants from Friedberg in 2021 perceived working with Athos and JSprit by expressing their level of agreement to positive statements about both approaches.

7.2.5 Results in terms of user satisfaction

This section presents the results of the survey questions related to user satisfaction. Just as in the original study, participants were asked to state their level of agreement to five positive statements on different language characteristics (see Section 7.1.5.1). Section 7.2.5.1 presents and discusses the results of the Friedberg study and Section 7.2.5.2 does the same for the results obtained from Wetzlar.

7.2.5.1 User satisfaction in Friedberg

In Figure 7.20 the degrees to which participants of the Friedberg study agreed to the different statements on both approaches are summarised. On a first note, the results where similarly positive for Athos as were those in the 2020 Study conducted

among students from Friedberg (see 7.1.5.1). Especially with regard to the perceived learnability, participants from Friedberg appear to be content with Athos. More than 4 out of 5 participants agreed that Athos could easily be learned (agreement ≈ 31 %, strong agreement ≈ 50 %). Only around 14 % of all participants expressed their doubts about this claim (disagreement ≈ 10.3 %, strong disagreement ≈ 3.4 %).

The vast majority of participants also ascribed a high readability to the Athos DSL. Nearly half of all participants strongly agreed that Athos models were easy to read and another 26 % stated their agreement to such a statement (agreement ≈ 26.8 %, strong agreement ≈ 48.2 %). The statements on easy program creation, efficient modelling and avoidance of bugs all were agreed to or even strongly agreed to by more than 60 %.

Also resembling the results from the 2020 study in Friedberg, participants did mostly not agree to the presented statements when these were applied to the JSprit approach. The best result for JSprit came with the statement on learnability. In this regard, around 14 % agreed and nearly 4.0 % strongly agreed to that statement. However, considerably more than 1 in 3 participants expressed their disagreement here (disagreement ≈ 29.3 %, strong disagreement ≈ 13.8 %). With the other a similarly high percentage refused to concur. The highest level of disagreement was observed for the statement on easy program creation. This statement was disagreed by nearly half of all participants (≈ 45.6 %) and strongly disagreed by another 21 %.

The presented results confirm the high levels of user satisfaction that were observed in the prior Friedberg study. In both surveys, participants clearly stated their preference for the Athos approach for the modelling of VRPs. JSprit as the baseline approach did not receive levels of appreciation that where anywhere near those received by Athos. Therefore, it can safely be concluded that user satisfaction with Athos was clearly superior to the user satisfaction brought about by JSprit.

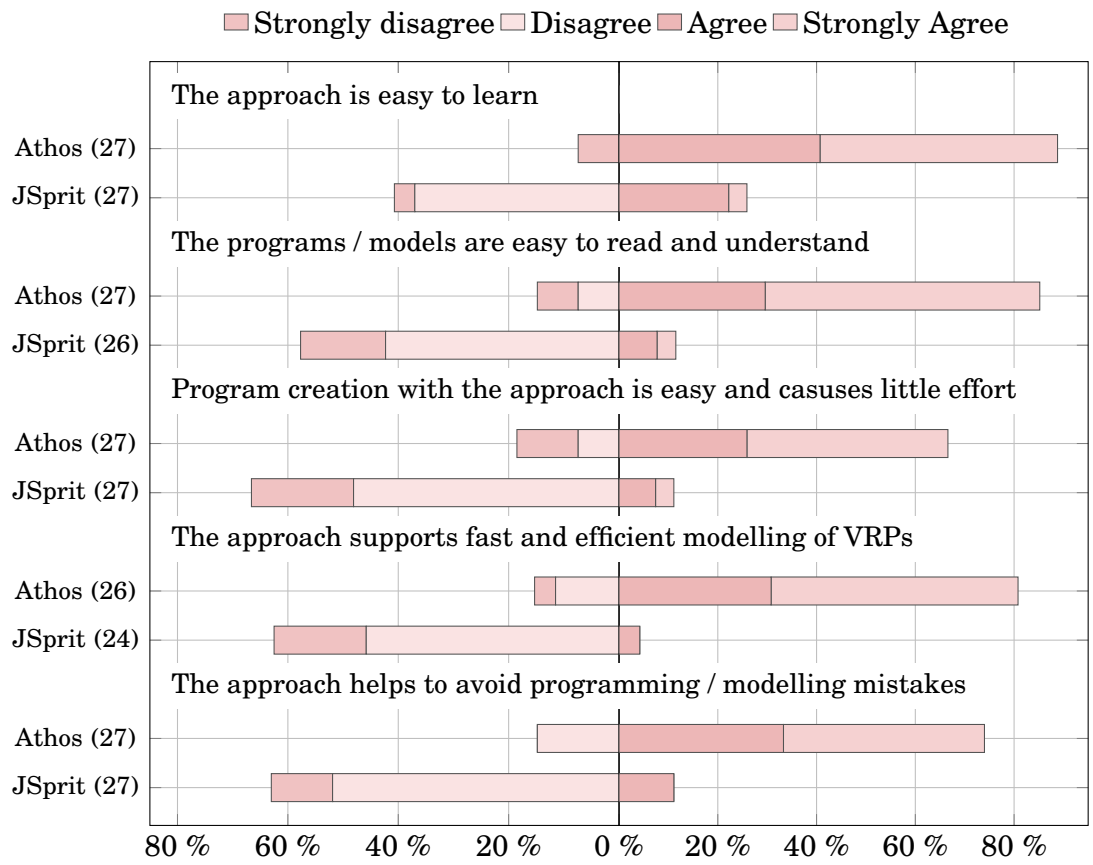


Figure 7.21: Overview on how participants from Wetzlar in 2021 perceived working with Athos and JSprit by expressing their level of agreement to positive statements about both approaches.

7.2.5.2 User satisfaction in Wetzlar

The levels of agreement participants from the Wetzlar study expressed are depicted in Figure 7.21. Similar to the 2020 study (see Section 7.1.5), for Athos these levels where somewhat higher among this study group than those observed in the Friedberg study group. However, the number of participants who concurred with the statement that Athos allows for easy program creation decreased moderately. Though this was the statement that found the least amount of agreement among participants, Athos still managed to convince 2 out of 3 participants in this regard (agreement $\approx$ 25.9 %, strong agreement $\approx$ 40.7 %).

Nearly 75 % expressed support for the claim that Athos helped them to avoid modelling mistakes. The other three statements were supported by upward of 80%. The strongest support came for the perceived ease with which Athos could be learned.

Around 40 % of all participants agreed and 48.1 % strongly agreed to the claim that Athos was easy to learn.

The JSprit approach, by contrast, was again not well appreciated by participants. Even the statement with the highest level of agreement was only supported by 1 in 4 participants. Around 23 % agreed that JSprit could easily be learned and 3.7 % were in strong agreement with this statement. On the negative side however, 37 % disagreed to this statement and 3.7 % indicated strong disagreement in this matter. The statement that met with most scepticism was the one that claimed that JSprit supported fast and efficient modelling. This claim was disagreed by almost half of the study population from Wetzlar (≈ 48.1 %) and strong disagreement was brought forward by 18.5 %.

These data provide additional evidence for claims on a high level of satisfaction that user experience when using Athos compared to alternative approaches. Participants from this study group clearly favour Athos over modelling with the JSprit alternative. While users appeared to generally appreciate modelling with Athos, they showed an especially high appreciation for the learnability and readability of Athos.

7.3 Summary and conclusion

This section provides a summary of the results obtained from the original and the replication study. The results are briefly discussed in terms of the statistical evidence found that regards how the approaches affected users' correctness and efficiency. It also considers the findings on the levels of satisfaction that users expressed for both approaches.

The findings of the original and the replication studies are summarised in Table 7.11 and Table 7.12. Both tables have a similar structure and show the approach of comparison taken (within-subjects vs. among-subjects) together with the year in which the tested data were collected (2020 vs 2021). Subsequent columns then show the number of tests that yielded a statistically significant advantage

for Athos (AT Sig) and the number of significance tests that proved a statistically significant advantage for JSprit (JS sig). The table also lists the number of those datasets that were not found to be statistically significant but in in which the observed average determined by the median was either in favour of Athos (Mdn A) or JSprit (Mdn JS).

Table 7.11: Summary of the results of all tests for correctness.

Correctness Test	A Sig	JS Sig	Mdn A	Mdn JS
Between-subjects Pts 2020 (Section 7.1.3.3)	1	0	2	1
Within-subjects Pts 2020 (Section 7.1.3.4)	3	0	0	1
Between-subjects Pts 2021 (Section 7.2.3.3)	2	0	2	0
Within-subjects Pts 2021 (Section 7.2.3.4)	4	0	0	0
Total	10	0	4	2

Table 7.11 summarises the number of statistical significance tests that analysed the scores obtained by study participants in both the original and the replication study. From 16 conducted statistical significance tests on the data obtained in the two studies, 10 yielded a statistically significant result in favour of Athos. No statistically significant evidence could be found in support of JSprit being the superior approach in terms of correctness. Of the remaining comparisons, the average user correctness was favoured Athos in 4 and JSpirt in 2 cases.

In summary, these data show clear support for Athos as the approach that enables superior results in terms of user correctness. The results that users produced when using Athos for tasks related to the learning about and perception of VRPTW models as well as modification of these models show that Athos can provide considerable support to users that enables them to achieve a higher level of correctness. For this reason, the first hypothesis H1 made at the beginning of the last chapter is to be *accepted*.

Table 7.12 shows a summary of the number of statistical significance tests that showed significantly better results for either of the two approaches. It also shows the number of comparisons in which no statistical significance was observed but in which the average efficiency was higher for either Athos or JSprit. From 16 comparisons,

Table 7.12: Summary of the results of all tests for efficiency.

Efficiency Test	A Sig	JS Sig	Mdn A	Mdn JS
Between-subjects PPM 2020 (Section 7.1.4.5)	2	0	2	0
Within-subjects PPM 2020 (Section 7.2.4.6)	3	1	0	0
Between-subjects PPM 2021 (Section 7.2.4.5)	3	0	1	0
Within-subjects PPM 2021 (Section 7.2.4.6)	3	0	1	0
Total	11	1	4	0

11 yieled statistically significant evidence that Athos enhanced the efficiency with which users learned, perceived and modified important part of models that represent VRPTWs. One of these tests showed a statistically significant advantage for JSprit users. All of the comparisons that were not statistically significant showed median values that were higher for Athos.

In total, these data show that Athos also enables users to learn, perceive and modify VRPTW models with significantly enhanced efficiency. The data show that Athos merits serious consideration for tasks that require efficient comprehension and creation of VRPTW models. The second hypothesis H2 made at the beginning of the last chapter hence is to be *accepted*.

Both studies also revealed substantially higher levels of user satisfaction participants perceived when working with Athos to solve the presented tasks. In all four studygroups, i.e. Friedberg 2020, Wetzlar 2020, Friedberg 2021, and Wetzlar 2021, the answers left no doubt that participants experienced Athos as the language that was easier to learn, read and comprehend, apply, and maintain.

7.4 Threats to validity

7.4.1 Threats to construct validity

All participants of the study knew a-priori that Athos was a language developed by the lecturer of the respective module in the course of which the evaluation was conducted. For this reason there is the threat that participants guessed the intended

hypothesis of the study was to prove that Athos was easier to use than the baseline approach. Under these circumstances, it cannot be ruled out that some participants might have felt the obligation to please the study conductor by working harder when using Athos as an approach. In order to mitigate this threat, participants were assured that data were collected in a completely anonymised fashion so that it was impossible to track any result back to a particular individual. Moreover, participants were encouraged to just answer the question for both approaches with their best effort. They were openly told that the results would be compared but that even a discovered superiority of the baseline approach would provide valuable scientific insights.

As the study was carried out remotely with an online survey tool, there was no reasonable way to ensure that participants actually used their time to work on the tasks of the question. While it would have been possible to ask participants to turn on their web cams while working participating in the study, this would have meant a severe intrusion into their privacy and be incommensurate with a voluntary participation in a DSL usability study. The fact that participants were sure to be unsupervised while solving the task poses the threat that different sources of distraction negatively affect the results of participants. It can't be ruled out that participant did not fully concentrate on the task due to other activities that they performed in parallel.

The best way to mitigate this threat was a clear and honest communication with participants. It was emphasised that participation would provide important scientific data on the language and that any non-standard behaviour would run the risk of distorting these data. Moreover, utmost care was taken to not pressure any student into participation. Instead, it was repeatedly emphasised that participation was on a voluntary basis and that those who only wished to inspect the questions (in preparation of the examination at the end of the semester) were free to do so. Students were told that they could opt out of any question if they did not intend to seriously try to solve the tasks presented in the questions. Based on experience, the vast majority of students respect clear and honest communication in conjunction with

the fair rules they were asked to adhere to. This way, only malevolent participants would actually derogate the value of the obtained data by the provision of answers without any interest in these answers being correct. The exclusion and inclusion criteria also somewhat mitigate these threatss cases are likely to be excluded if a participant answered the questions for one approach with considerably more focus than they spent on the other approach.

The threat to the validity that was detected concerns the creation of the questions. Even though considerable effort was put into the design of the questions with the online-survey tool, and in spite of a review of the questions by members of the research group, some faults in the formulation of the questions remained. In the second task of Q09ATAG, in the original study the code that was intended to be modified already included the correct solution. For this reason, the subtask was removed from the evaluation of the study and the first task was weighted 10 points on its own. To balance the evaluation, the second subtask of the respective JSprit question was also not used in the evaluation of the original study and the weight of the first task increased to 10 points. In the original study conducted in Friedberg, in Q03JSALL the same correct solution for the JSprit task could be spotted in the name of the class. It was still kept in the evaluation for three reasons. The first reason the question was not removed was that it biased the result of the study *against* Athos and favoured the baseline approach. The second reason was that the task still required a thorough read of the JSprit code and was easy to overlook. The third was that the mean score of all 73 participants considered in the study (including those who did not provide an answer for this question) was 3.97 and thus only slightly higher than the 3.33 points expectancy value that would have resulted if all participants had blindly guessed the answer (question type was multiple choice with three options and one correct answer).

7.4.2 Threats to internal validity

A major threat to the validity of the study is that it was conducted as a remote study via the Zoom meeting software. This offers various possibilitites to participants to answer the questions in an illicit way, e.g. by using IDEs, communicating with each other or using additional sources to answer the questions. This threat could not be mitigated on a technical level. However, the applied incentive can be considered of a way to address this problem: students did not receive a direct benefit for participation in the study. They were not awarded any points that would have altered the final mark they received for the respective study course module nor were they promised any other incentive that would have motivated them to take illicit measures in order to increase their score. On the contrary, the only communicated incentive was that they were told that the final exam would also encompass questions on vehicle-routing problems and that participation in the study could be regarded a bona fide training towards the question in the final exam. They were also told that application of additional measures would only hurt their own training effect. Finally, they were told that the data were for an important scientific study and that they would run the risk of damaging this study by not adhering to the specified rules. While this certainly does not ensure that there were not participants who infringed the defined rules of participation, it reduces the motivation to such behaviour to a minimum.

Another possible threat to the validity of the results stems from the applied inclusion and exclusion criteria. These criteria allowed for inclusion of results even in cases where these were significantly above or below results produced by participants of the same group. The study was intended to provide an impression on how various different users were able to cope with the language. For this reason, it was decided to only remove a case if the participant simply did not bother to answer the majority of the questions and in cases were the participant showed substantially different levels of effort with either of the two compared approaches. This way, the inclusion of results could skew the results of *between subjects* comparisons to the

detriment of the group the low performing participant was part of. The extent of this threat is mitigated by the application of a rank-based significance test. Through these tests the actual distance in points is dropped and only the low rank of the respective participant is considered. It is also to be noted here, that the actual number of evident outliers in the study was very low. With regard to *within-subjects* comparisons, the inclusion of low performers should not skew the study results as the inclusion exclusion criteria ensured that the difference of results produced for both approaches did not show unreasonable divergence.

Another threat to the internal validity of the study was that the threat that the difficulty of the tasks might be skewed. This way the results would be affected as the score for the approach with the simpler questions would most likely be higher. This threat was eliminated by taking utmost care to construct the questions in a way that they would be of identical complexity. for this reason, it was ensured that corresponding questions comprised the exact same number of entities, i.e. the same number of nodes, roads, etc. If possible, it was also ensured that the same number of possible answers was given. This was possible if a certain number of programs that contained mistakes were to be found by participants. In these cases, the number of options were identical for both approaches. However, in cases were participants should select a checkbox that indicated which line in the respective code contained an error, the number of presented checkboxes could not be made equal as JSprit programs are inherently more verbose than Athos programs. This however, is not a flaw in the construction of the question, but a characteristic of the language that this study intended to evaluate in terms of usability.

The study conductor (the author of this thesis) is another threat to the validity of the study as he was the one who introduced participants to both approaches. On the one hand, this way it could be ensured that differences in the way the approaches were presented to the students were identical for every study group. However, given that there was no pre-recorded script, the style and enthusiasm shown on the different days that the presentations were held are subject to variation which potentially affects the study results. A threat that arises from the fact that the lead

developer of Athos also conducted the introduction to both approaches is undeniable. Though the author put his best effort into presenting both approaches in the most objective way possible, there is no way to rule out that on an unconscious level a preference for Athos was signalled so this threat was not eliminated.

The time given to participants was another threat to the validity. This threat was eliminated by presenting both approaches on the same day and starting the presentation with JSprit so that – if only slightly – the time given to learn the approaches was slightly higher for JSprit and thus the bias was against Athos. As an additional measure of security, participants were also asked whether they felt that they were given more time for one of the two approaches. In 2020 91 of 101 participants stated that the time granted for both approaches was equal, 8 stated that they were given more time to learn the JSprit approach and 2 stated they were granted more time to get learn Athos. In the 2021 study, 68 of 84 participants stated that the time was identical and for each approach there were 8 participants that stated that Athos was given more time to be learned and another 8 that stated the same for JSprit.

On a similar note, the quality of the learning material could have posed a threat to the internal validity of the study. If the slides for one of the two approaches were made in a way that one approach could be learned easily while the other set of slides were hard to comprehend, this would greatly impact the results of the study. This threat was eliminated by having both sets of slides following the same structure in which the approaches were taught. Both sets dealt with the same problem and showed the same steps of how to model the problem. To further ensure that the results were not skewed by unequal learning material, participants were asked to state whether they felt that the learning material had similar quality. In the 2020 study, 88.7 % felt that the quality of the learning material was identical, 94 of 100 participants that provided an answer felt that the quality was identical, whereas 5 participants thought the Athos material to be superior and 1 participant preferred the quality of the JSprit slides. In the 2021 study, 69 of 86 participants stated that the quality was identical, 15 preferred the quality of the Athos slides and 2

considered the JSprit slides to be of higher quality. It would be interesting to further analyse as to what caused the higher percentage of participants preferring the Athos slides in the 2021 study since both slide sets were identical to the ones used in 2020.

A final threat to the internal validity concerns the the replication study in Friedberg. This study featured seven participants who had already participated a year earlier. This poses a threat to the internal validity of the study as they had considerable learning effects on their side. These participants were still included since – by chance – they were as equally distributed among the two subgroups as was possible for an odd number of students: four were assigned to the FbJf21 subgroup and three were part of the FbAf21 subgroup. Given the rank based statistical tests applied in the study and the applied filters of among-subject outliers, together with the four-to-three distribution ratio, the inclusion of these participants should – if at all – only slightly bias the between-subjects comparison.

7.4.3 Threats to external validity

The study population consisted entirely of students of THM from its campuses in Friedberg and Wetzlar. Though conducting of empirical evaluations among such populations of students is a common approach in the software engineering discipline (Sjoeberg *et al.*, 2005, p. 738), it poses a threat to the external validity of the study. It is therefore not claimed that the obtained results can be directly generalised to domain experts in the field of last mile logistics and transport planning and optimisation. However, by consideration of the academic background and the investigation on the prior programming knowledge of the study population, the study provides the necessary meta information of its population. This meta information can be applied in attempts to sensibly generalise the finding of the study to the target population of professionals in the field of transport optimisation.

Eighth Chapter

Conclusion

8.1 The Athos project in context of information systems design science

8.1.1 Design as an artefact (Guideline 1)

The first guideline postulated in the information systems (IS) design science framework of Hevner *et al.* (2004) demands that a design science research project yield an artefact that is both pioneering and useful. The artefact must either provide a solution to a relevant and hitherto unresolved problem, or it must improve on an existing solution to a relevant problem. Hevner *et al.* also point out that a valuable design science artefact does not necessarily have to be brought to perfection upon its first deployment, but may also be of prototypical nature with room for improvement that may well be brought about by ensuing research endeavours. The framework also emphasises, that the design scientist must provide a description for the artefact that facilitates an implementation and application in the intended domain.

Constructs are one specific type of artefact that a design science project can conceive. Constructs form languages or notations for the codification of a problem of interest together with its associated solution space. In this sense, the main contribution of this research project are the constructs that form and constitute Athos, that is, the constructs found in the meta-model of Athos as well as their concrete

representations which were presented in Section 4.1 and Section 4.3, respectively. The intended additional value to be brought about by the language was an increased usability in the simulation of traffic and transport related (optimisation) problems as they frequently occur in the business of LML.

It is important to note that Athos was not only conceived and defined in theory. Instead it was actually implemented by application of the language development framework Xtext[1], anchored in the foundational knowledge base for rigorous DSLs implementation. The Xtext framework generated the scaffold for (or in some cases entire) tools that built the language. The generator generated by the framework, for example, requires the implementation of the transformations to the respective target platform. The actual implementation of these transformations was discussed in Section Chapter 5. In terms of the IS design science framework, these transformations are *instantiations*. These instantiations demonstrate the feasibility of mapping Athos to agent-based simulation platforms.

8.1.2 Problem relevance (Guideline 2)

The second guideline defines the objective of design science research as the acquisition of skills and knowledge that put researchers in a position where they can develop novel solutions for a given problem domain. The necessity for such novel tools in the domain of simulation modelling has been described by Taillandier *et al.* (2019b) who emphasize the need for modelling tools that enable 'participatory modelling'. By this, the authors mean a modelling approach that views domain experts as active creators of the models that drive the simulation studies.

The second guideline also stresses the importance of an active community behind the artefact produced. Though Athos has not yet been presented to a community of field practitioners, the formal evaluation presented and discussed in this thesis clearly indicates that the DSL is on the right way by being comparatively easy to learn and apply. This way, the language is highly likely to meet the approval of

[1] https://www.eclipse.org/Xtext/

experts with scarce programming knowledge but plenty of valuable know-how in the domain of traffic and transport optimisation.Athos then becomes a highly relevant artefact as such persons can rely on its high usability and expressiveness.

8.1.3 Design evaluation (Guideline 3)

Hevner *et al.* (2004) stress the importance of a rigorous evaluation of the designed artefact. In the preparation of the evaluation, scientists have to select (and probably adapt) appropriate evaluation methodologies together with expressive measures from the knowledge base. The authors also list several design evaluation methods together with a subsuming category. They note, that evaluation methods, applied measures, and the evaluated artefact have to be compatible in order to conduct high-quality evaluation research.

As was shown in Chapter 6, evidence for the utility created by Athos was provided by conducting a *controlled experiment*. In this experiment, Athos was compared to JSprit[2], an alternative approach to model and solve VRPs based on the Java GPL. The applied evaluation methodology was based on the work of Kosar *et al.* (2010), Kosar *et al.* (2012) and Kosar *et al.* (2018) and additional planned and executed under consideration of the process laid out in the Usa-DSL evaluation framework (Poltronieri *et al.*, 2018). Athos and JSprit were compared in terms of how they affected participants' *efficiency* and *effectiveness* in *learning*, *perceiving* and *evolving* the languages and models/programs written in the respective language.

The performed evaluation also considered qualitative aspects of the languages. Participants were asked for an assessment on the complexity of learning and applying the respective languages and whether they felt that the respective language helped them to avoid programming mistakes. Participants were also given the chance to state their general opinion on both approaches. This way, the evaluation

[2] `https://jsprit.github.io/`

also complies with the request to also evaluate the *style* of the designed artefact. Especially in the open-ended questions participants expressed their opinion on what can be considered the style of the two compared languages.

The comparative approach showed that Athos is a *necessary* artefact as participants clearly stated their preference on using Athos instead of JSprit. Moreover, especially participants with only basic Java experience were highly more effective and efficient when using Athos instead of JSprit. According to Hevner *et al.* (2004), the creation of new or improved artefacts together with a formal proof of their (higher) utility is at the heart of design science.

8.1.4 Research contributions (Guideline 4)

Hevner *et al.* (2004) state that there are three different ways in which a design science research project can make a significant contribution. Firstly, for many design science projects, the contribution is the utility the designed artefact provides to the targeted community of researchers and or practitioners. Secondly, a scientific contribution can be made by means of an enhancement of the foundations of the knowledge base that is used by researchers for the creation of relevant artefacts. And thirdly, innovative application and enhancement of evaluation methodologies from the knowledge base also constitutes a valuable contribution to the scientific body of knowledge.

This research project contributes in each of these three ways. First of all, *Athos as a holistic system* (i.e. the language and its supporting development tools together with its generator) can be proficiently applied by a community of traffic and transport researchers and practitioners. In addition to that, Athos can also be regarded as a *construct* that scientists can use to create innovative simulation *instantiations*. Especially its built-in *extension mechanism* enables researchers to leverage Athos in the creation of novel simulation artefacts or in the development and evaluation of routing optimisation algorithms (Hoffmann *et al.*, 2020). Finally, as was discussed in Chapter 6, Athos was subjected to a rigorous evaluation approach based on

established evaluation methodologies that were slightly enhanced in order to obtain proper evidence for the utility provided by Athos.

8.1.5 Research rigor (Guideline 5)

This research project adhered to the fifth guideline ('research rigour') by the application of well developed and well understood technologies and approaches for the development, implementation and evaluation of DSLs. For the development and implementation of the language, the well-known Xtext language development framework[3] was used.

In the iterative and incremental DSL development process, guidelines presented in (Karsai *et al.*, 2014) were considered in the implementation of the language. For example, Karsai *et al.* (2014) advise in Guideline 22 to take an convention over configuration approach, i.e. assume sensible defaults that relieve end users from having to specify every single detail of the model. This guideline was followed on multiple occasions, e.g. edges are assumed to be undirected unless explicitly specified as directed arcs, the first behaviour specified for an agent is always the behavioural entry state of an agent, etc. Another example in that direction is that Athos takes a computationally independent approach, i.e. users do not need to specify how agents calculate an optimal tour for a set of nodes – if the model does not explicitly specify a specific optimisation algorithm or specific parameters for a given algorithm, sensible default values are used (Hoffmann *et al.*, 2018b).

The 21st guideline presented by Karsai *et al.* (2014) recommends to design a language's concrete syntax in a way that a given style is used consistently throughout the entire language. To also provide a descriptive concrete syntax (Guideline 15) as well as organisational structure (Guideline 19), the following rule regarding the usage of commas in Athos models was introduced: whenever elements are listed in an Athos model, commas can optionally be used. A recommendation to users is to

[3] `https://www.eclipse.org/Xtext/`

either consistently use commas for the separation of listed elements or not at all. However, this recommendation is not enforced by the validator.

As was discussed in Section 8.1.3, an elaborate controlled experiment based on the knowledge shared by scientists who are well-recognised for their expertise in language evaluation Kosar *et al.* (2010), Kosar *et al.* (2012) and Kosar *et al.* (2018) and Poltronieri *et al.* (2018) were followed. Their published works greatly affected the general nature of the performed evaluation (a controlled experiment that compared Athos and and alternative approach), the language aspects considered in the evaluation (learnability, perceivability, evolvability), the metrics used to determine the languages' performance in the different aspects to consider (effectiveness and efficiency), and also the way data for these metrics was obtained (different questions aiming at different aspects of the language in different ways). By relying on well-known evaluation approaches, it was ensured that rigorous results were obtained that allowed a qualified statement on the languages' provided utility.

However, it is also important to note, that an exaggerated concentration on rigour may hamper the relevance of the research Hevner *et al.*, 2004. The authors emphasise that whether a design science project succeeds or fails is determined by the ability of the researchers to beneficially apply the knowledge base throughout the project. Similarly, (Karsai *et al.*, 2014) also note that their presented guidelines are not to be understood as an exact recipe to follow in the pursuit of a perfect language. On the contrary, the authors stress that there are trade-offs between some of the guidelines and that it is the task of the language developer to carefully consider the given domain and application context to determine which guideline takes precedence over the other(s).

8.1.6 Design as a search process (Guideline 6)

In the sixth guideline of their design science framework, Hevner *et al.* (2004) describe how design science often has to apply an iterative incremental approach that addresses a complex problem by breaking it down into and solving its constituent

sub-problems. Though it may be necessary to recursively break down and solve the sub-problems, at some point a sensible starting point will be found. As is emphasised by the authors, this process might result in a starting-point problem whose solution contributes only little to the underlying real-world problem. However, through iterative incrementation of the problem scope and development of appropriate solutions a continuous progress is achieved until finally the original problem is solved.

This is what has been done in the course of this research project. The domain of traffic and transport simulation and optimisation was broken down into several sub-problems that were either analysed and solved or broken down further until a starting point was found. The first capabilities of Athos were the representation of networks in which sources of agents of different types could be defined and edges could be associated with mathematical functions that determined the time an agent required to traverse the respective edge depending on the traffic on that edge. After that, the TSP as the first member of the family of VRPs was addressed followed by the VRPTW. Each of these steps comprised several sub-problems, like the definition of appropriate language elements to represent the problem, the definition of suitable transformation to the NetLogo (or Repast) platform, or the implementation of algorithms to obtain (near) optimal solutions for modelled problems. Through a process of iteratively increasing the languages functionality, the DSL and its supporting tools were brought to a state of maturity.

Hevner *et al.* (2004) also point out that having a finished design science artefact which provides a solution to a targeted problems, poses the challenge of an informed assessment on how well the artefact actually handles the problem. In order to evaluate the quality of the provided solution, the authors propose a comparison between the developed artefact and alternative solutions developed by recognised experts in the respective field. This is what has been done for Athos. Several usability aspects of Athos (e.g. effectiveness and efficiency) were compared to those offered by the JSprit[4] application library for the Java language. As a result of this comparison it could be shown that especially participants who had little programming exper-

[4] `https://jsprit.github.io/`

ience achieved far better results in effectiveness and efficiency when using Athos than when using the application library. Moreover, from programming novices to experienced programmers, most participants preferred using Athos over JSprit.

Even at this point, Athos is not a completely finished modelling system. There are still several iterations possible to increase the power of Athos. Currently, an additional viewer plug-in is being developed. This plug-in visualises the textually modelled network. Moreover, it is also possible to edit the network in the plug-in. The plug-in viewer is synchronised with the Athos editor so that changes in one of both are recognised and adopted in the other. A first prototype of this plug-in has been created by a student of THM in his Master's project. However, the tool still needs some bug fixes and modifications before it can be fully integrated into the Athos environment. Though Athos possesses an extension mechanism so that additional vehicle routing problems can be represented in the language and solved by appropriate algorithms, the language should be extended in order to natively support additional problems of the VRP family. Another way to improve the Athos might be the development of new transformation that create simulations for an even more powerful simulation platform that NetLogo. The Gama platform appears to be a promising platform that might offer even deeper insights into the modelled problems than NetLogo does, due to its advanced exploration capabilities.

8.1.7 Communication of research (Guideline 7)

The design science research framework demands that research artefacts and evaluation results be communicated to both experts in the technological aspects of the research and those concerned with the managerial implications of the designed artefact. This previous section described how new functionality was added to Athos in an iterative incremental way. With each iteration, Athos reached a new level of maturity and the new functionality was presented at pertinent conferences and described in the respective proceedings. The detailed results of the elaborate empirical evaluation study were submitted to the Empirical Software Engineering journal.

All publications should be of interest to both technical and management experts. Though most articles featured a technical explanation of one or several new functionalities implemented in the course of the respective development cycle, they also featured a small case study with a focus on how to use the newly introduced features. The papers also discussed the respective benefits provided by the respective feature. This way, researchers in the technicalities could learn on how features were implemented whereas application oriented readers gained insight on how the feature could be used.

8.2 Summary and addressed research questions

Athos as an innovative construct illustrated the properties that a DSL which is to enable users to correctly and efficiently produce models of agent-based VRPs must have (RQ 1). It was shown which elements from the domain of (academic) vehicle-routing problem were most relevant (RQ 1.1) so that they needed to be mapped to the abstract syntax of the language (RQ 1.2). It was also shown how the abstract syntax was to be devised to allow for a representation of dynamic agent behaviour (RQ 1.3). For these behaviour elements it was explained how and to what target elements in the solution domain they needed to be mapped in order to bring about the intended execution semantics (RQ 1.6) and (RQ 2.1). This was also done for the static elements that form the environment within which the agents act and interact (RQ 1.4) and (RQ 1.6) .

To support modellers in the development of semantically correct models, an example for a concise yet expressive concrete syntax was illustrated (RQ 1.5). To further support ensure that the created models are valid an initial set of constraints that form the static semantics of the language was built (RQ 1.4). It was also demonstrated how methods can be implemented that allow to derive optimal solutions and how these could be linked to the language environment and simulation environment (RQ 2.2).

The final part of this thesis gave a summary of a method to reveal and compare innate usability characteristics of different modelling approaches (RQ 3.1) and how to quantitatively represent the disclosed differences (RQ 3.2). From the quantified differences a set of claims concerning increased correctness, efficiency and user satisfaction associated with the application of Athos was shown to be supported by substantial statistic evidence (RQ 3.3).

8.3 Future work

As was already mentioned at the beginning of this thesis, the Athos research project made two major contributions: a novel DSL for the specification of traffic and transport simulation and optimisation scenarios and a rigorous empirical evaluation of this language. According two these two major contributions, the future work can be split into two distinct stacks of work that has to be done:

- Future work on the Athos language
  - **Improvement of the language's expressiveness by introduction of new language elements** supported by the generator and the targeted simulation platform.
  - **Improvement of the language's extension mechanisms** by making the language extensible without the need to access and recompile the meta-model definition of the language.
  - **Introduction of language elements for user-defined constraints** for problem solutions.
  - **Introduction of additional target platform** to which Athos models can be transformed.
- Future empirical work centred around Athos
  - **Comparative evaluation studies** to evaluate (new or improved) language elements like metrics or dynamic VRPs.

- **Effects that several ways to achieve the same goal** in a language have on users' satisfaction and confidence with a language.
- **Effects of modern browser-based tooling** on the usability of the language.
- **Evaluative application by real domain-experts** to increase the generalisability to the targeted user group.

These aspects will be discussed in more detail in the following sections.

8.3.1 Expressiveness of the language to be improved

The current traffic flow model applied in the generated agent-based simulations is a plain and straightforward model to simulate the movements of entities in a network. On the one hand, it considers every agent in the network an individual and thus *microscopic level*, on the other hand, the way agents affect one another is mainly dealt with at a *macro level* (see, e.g., (Barceló, 2010, p. 15)). Though the simplicity of the applied model brings about benefits in terms of simulation performance, it must be assumed that there exist different emergent real-world phenomena that cannot sufficiently be modelled with the applied traffic flow model. It thus necessitates further investigation how the language and the transformations can be extended to dynamically allow a finer-grained simulation approach. It appears desirable to enhance Athos in a way that the level of granularity can be implicitly defined by the modeller through the language elements applied in a model.

This requires the language to be extended by behavioural language elements that allow a fine-grained description of an agent's general behaviour. These meta-model elements could, for example, allow expressions of *circumstances* under which the agent moves a given percentage faster or slower than the average travelling speed on the current edge. Evidently, these extensions of the language require an adoption of the generated traffic-flow model that needs to be adapted accordingly. Attempts in these directions have been made but have not yet reached a level of maturity that allowed them to be integrated into the main development branch.

The concrete syntax for the metrics defined in the metrics section of an Athos program will be further refined. This requires additional research into what syntactical structures allow to concisely convey the intended semantics of the defined metric and can conveniently be specified by the user. For this, additional studies (e.g. field studies or additional controlled experiments (Hevner *et al.*, 2004, p. 86)) are required to determine how users can be best supported in the definition of the metrics to monitor throughout a simulation. In order to be able to perform a comparative controlled experiment like those presented in this thesis, an appropriate baseline language to which Athos can be compared must be found. JSprit might well be an appropriate language with regard to the evaluation of a language's metric definition capabilities. However, JSprit itself focusses on static problems and will thus not be a suitable baseline approach when it comes to the evaluation of dynamic aspects of Athos.

8.3.2 Extensibility of the language to be matured

In Hoffmann *et al.*, 2019a it was presented how Athos offers a mechanism that allows users to include additional information into Athos models and provide these to Java algorithms. This mechanism was supposed to provide the possibility of user extensions (Atkinson and Kuhne, 2003, p. 37). While this mechanism can be helpful in certain circumstances, it reduces the readability and understandability of Athos models.

Future research will investigate on how additional information can be included into Athos models in a user friendly way. Currently, Athos allows the definition of additional agent and edge attributes. Future versions should enable users to extend any aspect of an Athos model to their needs. However, it has to be taken care that these extensions are implemented in a way that preserves the domain-specific nature of Athos programs.

8.3.3 Extension of the static semantics of the language

In Section 4.2, Table 4.2 on 111 gave an overview on the currently active static semantics for Athos. As can be seen from the table, there is still room for improvement with regard to this languague apsect. Especially concerning the definition of agent behaviour, the language currently lacks a strong set of constraints that prevent users from time and cost intensive modelling mistakes. For example, there will have to be constrains that automatically recognise if an agent is modelled to perform a delivery without the agent having loaded the necessary cargo. With regard to the network, there must be a constraint that ensures that a customer node is connected to the network so that the agent can serve the customer.

While modelling mistakes like the aforementioned links are highly likely to occur, it will also be an interesting task to find out which constraints are actually helpful for users and which constraints will be considered annoying. This might pose another opportunity for a controlled experiment targeted at finding out which preventable modelling mistakes users make most often and for which they would appreciate to be guided by the static semantics of the language.

8.3.4 Integration of user-definable solution constraints

While Section 4.2 showed how pre-defined constraints can help users to avoid modelling mistakes, it would also be highly interesting and helpful to add language elements for the specification of user-defined constraints. These are constraints that target the solution of the modelled VRPs. For example, Athos could offer the syntactical elements to specify that in the optimal valid solution vehicle A must visit customer a, b, and c directly after one another and that customer z may not be in the same tour as customer b. This would increase the practical applicability of the language as domain-experts then can introduce real-world constraints in the academic solutions providers.

8.3.5 Additional target platform to be addressed

NetLogo currently is the primary target platform for Athos. It might, however, be interesting to implement transformations to alternative platforms as well. While NetLogo is an appropriate target platform for the current state of Athos, considering aspects as runtime performance and scalability are likely to necessitate a target platform suitable for very large numbers of nodes, edges and most importantly agents. Besides, targeting two alternative platform can even increase confidence in the results drawn from generated simulations if they are identical on every targeted simulation platform (cf. (Hoffmann *et al.*, 2018b)).

8.3.6 Empirical studies to be conducted

8.3.6.1 Metrics and dynamic aspects of the language

The empirical evaluation presented in this thesis focused on the static modelling of VRPTWs. Future evaluations are required to focus on Athos' capability to model dynamic problems by inclusion of questions that feature tasks in which additional agent-sprouting mechanisms are to be used. These tasks then should also encompass the modelling of complex agent behaviour and behaviour transitions.

As was already mentioned it is also necessary to evaluate the currently present metric definitions. This is necessary to obtain valuable information on how to extend and further develop Athos' capabilities in this regard. This could be achieved through the inclusion of tasks in which users have to formulate the correct metrics, i.e. the correct aggregation level, calculation trigger, and calculation expressions (see Section 3.4.2.2) to track various data on phenomena occurring in a simulation.

There is also additional empirical research required that evaluates how the Athos' IDE plug-in affects its usability. To this end, it would also be highly interesting to gain deeper insight into the extent to which this plug-in affects the observed

Listing 8.1: An alternative source and demand definition.

```
 8 network
 9   nodes
10     n0 at (3, -4) // isDepot foo sprouts bar customers n1, n2, n3 at 0 latestTime 300
11     n1 at (-6, 8) // hasDemand foo units 15 earliestTime 30 latestTime 45 serviceTime 3
       ⋮
25   sources
26     n0 isDepot product sprouts vehicles customers n1, n2, n3 at 0 latestTime 300
27   demands
28     n2  hasDemand product units 10 earliestTime 30 latestTime 60 serviceTime 7
```

correctness, efficiency and user satisfaction. Similar additions to the original study presented by Kosar *et al.* (2010) were made in the replication study presented by Kosar *et al.* (2018).

8.3.6.2 Effects of alternative styles on the usability of Athos

The current version of Athos features some language parts that can be defined in several different ways. For example, Section 4.1.3 discussed how an agent type can be defined in an ad-hoc manner at a position in a program where an agent type is required (in-place agent type specification) or in the agent section of an Athos program where it can be referenced from places in the program where an agent type is required. Another example was discussed in Section 4.3.1 where it was shown that there are two alternative concrete ways to specify a time window for a demand note. A third example is depicted in the definition of source nodes and demand for nodes in a network: these are normally defined together with the specification of the node in the network section of the program (see Figure 3.6). An alternative way is depicted in Listing 8.1. The program in this listing is a modification of the first example from the introduction. The comments show the original approach used for the definition of sources and demands. The alternative is the definition of a *source* and *demand section* in which nodes from the network section can be referenced to define them as source nodes or demand nodes. This alternative stems from prior versions of the language and was kept for user who might consider this separation of concern preferable.

An interesting question in this regard is whether these alternative approaches support or hamper users in learning how to define correct programs. While at first

it might appear that an alternative will certainly not negatively affect learnability. However, it might well be possible that too many design alternatives can cause confusion among language learners. It might also prevent them from memorising one approach to model an aspect as they to remember both valid approaches (and end up forgetting both). Another difficulty here is that it might negatively affect communication between different language users in case both prefer different approaches and are unable to reach an agreement on which style to apply in the future. In the worst case, such a situation might even risk the benefit of improved communication between software and domain experts (for whom the chance of having dissenting style preference might be especially high). Another aspect in this regard is the support provided by auto-completion tools which might also be weakened in case that too many suggestions clutter the IDE at a particular point in the program.

For these reasons it seems worthwhile to empirically investigate on the effects the existence of synonymous modelling approaches has on the usability of a language. With such an investigation it would be possible to answer the question of a) whether such alternatives have an innate negative effect on the learnability of a language, and b) whether this effect is extenuated or even exacerbated by the application of modern IDEs that provide code completion mechanics. Finally, it would be highly interesting to investigate on possibly negative effects on cooperation. To this end, an experiment could be conducted, in which two different groups are introduced to the language. Both groups are taught different approaches. In a controlled experiment, a given modelling task is to be solved for which participants have to work in teams with some teams being formed of members from the same group and some teams with members from the two different groups. It will then be investigated whether the teams formed of members from the same learning group perform better than those formed of participants from two different learning groups.

8.3.6.3 Effects of a graphical web-editor on the usability of the language

As of writing this thesis, a student from THM is currently working on his Master's thesis in which he aims at the creation of a web-editor with graphical modelling

elements based on the Athos meta-model. This approach is similar to the one presented by Taillandier *et al.* (2019a) in which GAMA is based on the meta-model of the textual GAML language. The thesis also aims at carving-out an approach to formally evaluate the benefits brought about by the graphical modelling capabilities. The thesis is in a very early stage and it will have to be seen to what extend Athos will benefit from the achieved results.

8.3.7 Application by domain experts in the field

The two controlled experiments conducted in the scope of this thesis were performed with a population of students. While this is in fact a common practice found in most controlled experiments conducted in the domain of empirical software engineering (Sjoeberg *et al.*, 2005), Athos will have to be tried and tested by real domain experts from the field of traffic simulation and transport optimisation. Being able to gather a sample population of at least 10 to 15 domain experts from all over the world would increase the generalisability of obtained results to the targeted user group of Athos. As domain experts cannot be expected to have too much free time at their disposal, the survey to which real domain experts will be invited will have to be considerably shorter than the ones this thesis reported on. Most likely, these surveys will also have to be conducted online. As was mentioned in Section 8.3.6.3, there is currently a student working on his Master's thesis which aims at developing a concept for a browser-based survey on the usability of Athos among experts from the domain of traffic and transportation optimisation. However, the student just recently set out to develop the necessary tools to make Athos operate in a browser-based environment which may then be modified in order to conduct online experiments.

Bibliography

Alaca, Omer Faruk, Baris Tekin Tezel, Moharram Challenger, Miguel Goulão, Vasco Amaral and Geylani Kardas (2021). 'AgentDSM-Eval: A framework for the evaluation of domain-specific modeling languages for multi-agent systems'. In: *Computer Standards & Interfaces* 76, p. 103513. DOI: 10.1016/j.csi.2021.103513.

Albuquerque, Diego, Bruno Cafeo, Alessandro Garcia, Simone Barbosa, Silvia Abrahão and António Ribeiro (2015). 'Quantifying usability of domain-specific languages: An empirical study on software maintenance'. In: *Journal of Systems and Software* 101, pp. 245–259. DOI: 10.1016/j.jss.2014.11.051.

All the Research (2021). *Last Mile Delivery Market: by Vehicle (Light Duty Vehicle, Medium Duty Vehicle, Heavy Duty Vehicle), by Cargo (Dry Goods, Postal, Liquid Goods), by End User (Chemical, Pharmaceutical and Healthcare, FMCG, E-Commerce) by Region (North America, Europe, Asia-pacific, Rest of the World): Global Forecasts 2021 To 2027*. Ed. by All the Research. URL: https://www.alltheresearch.com/report/755/last-mile-delivery-market (visited on 17/09/2021).

Atkinson, C. and T. Kuhne (2003). 'Model-driven development: a metamodeling foundation'. In: *IEEE Software* 20.5, pp. 36–41. DOI: 10.1109/MS.2003.1231149.

Axelrod, Robert (1997). 'Advancing the art of simulation in the social sciences'. In: *Simulating Social Phenomena*. Ed. by Rosaria Conte, Rainer Hegselmann and Pietro Terna. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 21–40. DOI: 10.1007/978-3-662-03366-1_2.

Baldacci, Roberto, Maria Battarra and Daniele Vigo (2008). 'Routing a Heterogeneous Fleet of Vehicles'. In: *The Vehicle Routing Problem: Latest Advances and New Challenges*. Ed. by Bruce Golden, S. Raghavan and Edward Wasil. Vol. 43. Operations Research/Computer Science Interfaces. Boston, MA: Springer US, pp. 3–27. DOI: 10.1007/978-0-387-77778-8_1.

Ballare, Sudheer and Jane Lin (2020). 'Investigating the use of microhubs and crowdshipping for last mile delivery'. In: *Transportation Research Procedia* 46.6, pp. 277–284. DOI: 10.1016/j.trpro.2020.03.191.

Balmer, Michael, Marcel Rieser, Konrad Meister, David Charypar, Nicolas Lefebvre and Kai Nagel (2009). 'MATSim-T'. In: *Multi-Agent Systems for Traffic and Transportation Engineering*. Ed. by J. Davim, Ana Bazzan and Franziska Klügl. Advances in Mechatronics and Mechanical Engineering. IGI Global, pp. 57–78. DOI: 10.4018/978-1-60566-226-8.ch003.

Bandini, Stefania, Sara Manzoni and Giuseppe Vizzari (2009). 'Agent Based Modeling and Simulation: An Informatics Perspective'. In: *Journal of Artificial Societies and Social Simulation* 12.4, p. 4. ISSN: 1460-7425. URL: http://jasss.soc.surrey.ac.uk/12/4/4.html.

Barceló, Jaume (2010). 'Models, Traffic Models, Simulation, and Traffic Simulation'. In: *Fundamentals of Traffic Simulation*. Ed. by Jaume Barceló. New York, NY: Springer New York, pp. 1–62. DOI: 10.1007/978-1-4419-6142-6_1.

Barisic, Ankica, Vasco Amaral and Miguel Goulao (2012a). 'Usability Evaluation of Domain-Specific Languages'. In: *2012 Eighth International Conference on the Quality of Information and Communications Technology (QUATIC)*. Ed. by João Pascoal Faria. Piscataway, NJ: IEEE, pp. 342–347. DOI: 10.1109/QUATIC.2012.63.

Barisic, Ankica, Vasco Amaral, Miguel Goulao and Bruno Barroca (2012b). 'How to reach a usable DSL? Moving toward a Systematic Evaluation: Electronic Communications of the EASST, Volume 50: Multi-Paradigm Modeling 2011'. In: *Electronic Communication of The European Association of Software Science and Technology* 50. DOI: 10.14279/tuj.eceasst.50.741.

Barišic, Ankica, Vasco Amaral, Miguel Goulao and Bruno Barroca (2011). 'Quality in use of dsls: Current evaluation methods'. In: *Proceedings of the 3rd INForum-Simpósio de Informática (INForum2011)*.

Barišic, Ankica, Vasco Amaral, Miguel Goulão and Bruno Barroca (2014). 'Evaluating the usability of domain-specific languages'. In: *Software Design and Development: Concepts, Methodologies, Tools, and Applications*. IGI Global, pp. 2120–2141. DOI: 10.4018/978-1-4666-4301-7.ch098.

Bazzan, Ana L. C. and Franziska Klügl (2014). 'A review on agent-based technology for traffic and transportation'. In: *The Knowledge Engineering Review* 29.3, pp. 375–403. DOI: 10.1017/S0269888913000118.

Bazzan, Ana L.C., Milton Heinen and Maicon de Brito do Amarante (2015). 'ITSUMO: An Agent-Based Simulator for Intelligent Transportation Systems'. In: *Advances in Artificial Transportation Systems and Simulation*. Boston: Academic Press, pp. 1–20. DOI: 10.1016/B978-0-12-397041-1.00001-7.

Bettini, Lorenzo (2016). *Implementing domain-specific languages with Xtext and Xtend: Learn how to implement a DSL with Xtext and Xtend using easy-to-understand examples and best practices*. Community experience distilled. Birmingham: Packt Publishing. ISBN: 9781786464965. URL: http://lib.myilibrary.com/detail.asp?ID=951754.

Borenstein, David Bruce (2015). 'Nanoverse: A constraints-based declarative framework for rapid agent-based modeling'. In: *Proceedings of the 2015 Winter Simulation Conference*. Ed. by Levent Yilmaz. IEEE, pp. 206–217. DOI: 10.1109/WSC.2015.7408165.

Borshchev, Andrei and Alexei Filippov (2004). 'From system dynamics and discrete event to practical agent based modeling: reasons, techniques, tools'. In: *Proceedings of the 22nd international conference of the system dynamics society*. Vol. 22, pp. 25–29.

Boysen, Nils, Stefan Fedtke and Stefan Schwerdfeger (2021). 'Last-mile delivery concepts: a survey from an operational research perspective'. In: *OR Spectrum* 43.1, pp. 1–58. DOI: 10.1007/s00291-020-00607-8.

Brambilla, Marco, Jordi Cabot and Manuel Wimmer (2012). *Model-driven software engineering in practice*. Vol. #1. Synthesis lectures on software engineering. [San Rafael, Calif.]: Morgan & Claypool. ISBN: 9781608458820.

Bratman, Michael (1987). *Intention, plans, and practical reason*. Cambridge, Mass. Harvard University Press. ISBN: 0674458184.

Bresciani, Paolo, Anna Perini, Paolo Giorgini, Fausto Giunchiglia and John Mylopoulos (2004). 'Tropos: An Agent-Oriented Software Development Methodology'. In: *Autonomous Agents and Multi-Agent Systems* 8.3, pp. 203–236. DOI: 10.1023/B:AGNT.0000018806.20944.ef.

BSI (2001). *Information technology. Software product quality*. London. DOI: 10.3403/02304484.

BSI, (2011). *Systems and software engineering. Systems and software quality requirements and evaluation (SQuaRE). System and software quality models*. London. DOI: 10.3403/30215101.

Busch, F. and G. Kruse (2001). 'MOTION for SITRAFFIC - a modern approach to urban traffic control'. In: *ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No.01TH8585)*, pp. 61–64. DOI: 10.1109/ITSC.2001.948630.

Cabot, Jordi and Martin Gogolla (2012a). 'Object Constraint Language (OCL): A Definitive Guide'. In: *Formal Methods for Model-Driven Engineering: 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2012, Bertinoro, Italy, June 18-23, 2012. Advanced Lectures*. Ed. by Marco Bernardo, Vittorio Cortellessa and Alfonso Pierantonio. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 58–90. DOI: 10.1007/978-3-642-30982-3_3.

Cabot, Jordi and Martin Gogolla (2012b). 'Object Constraint Language (OCL): A Definitive Guide'. In: *Formal Methods for Model-Driven Engineering: 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2012, Bertinoro, Italy, June 18-23, 2012. Advanced Lectures*. Ed. by Marco Bernardo, Vittorio Cortellessa and Alfonso Pierantonio.

Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 58–90. DOI: 10.1007/978-3-642-30982-3_3.

Cachero, Cristina, Santiago Meliá and Jesús M. Hermida (2019). 'Impact of model notations on the productivity of domain modelling: An empirical study'. In: *Information and Software Technology* 108, pp. 78–87. DOI: 10.1016/j.infsof.2018.12.005.

Carley, K. M., D. B. Fridsma, E. Casman, A. Yahja, N. Altman, Li-Chiou Chen, B. Kaminsky and D. Nave (2006). 'BioWar: Scalable agent-based model of bioattacks'. In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 36.2, pp. 252–265. DOI: 10.1109/TSMCA.2005.851291.

Challenger, Moharram, Geylani Kardas and Bedir Tekinerdogan (2016). 'A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems'. In: *Software Quality Journal* 24.3, pp. 755–795. DOI: 10.1007/s11219-015-9291-5.

Chan, Wai Kin Victor, Young-Jun Son and Charles M. Macal (2010). 'Agent-based simulation tutorial - simulation of emergent behavior and differences between agent-based simulation and discrete-event simulation'. In: *Proceedings of the 2010 Winter Simulation Conference*. IEEE, pp. 135–150. DOI: 10.1109/WSC.2010.5679168.

Clark, Jim and Gene Daigle (1997). 'The importance of simulation techniques in ITS research and analysis'. In: *Proceedings of the 29th conference on Winter simulation - WSC '97*. Ed. by Sigrún Andradóttir, Kevin J. Healy, David H. Withers and Barry L. Nelson. New York, New York, USA: ACM Press, pp. 1236–1243. DOI: 10.1145/268437.268766.

Cleaveland, J. C. (1988). 'Building application generators'. In: *IEEE Software* 5.4, pp. 25–33. DOI: 10.1109/52.17799.

Cordeau, Jean-François, Gilbert Laporte, Martin W.P. Savelsbergh and Daniele Vigo (2007). 'Chapter 6 Vehicle Routing'. In: *Transportation*. Vol. 14. Handbooks in Operations Research and Management Science. Elsevier, pp. 367–428. DOI: 10.1016/S0927-0507(06)14006-2.

Cuadrado, Jesús Sánchez, Javier Luis Cánovas Izquierdo and Jesús Garc\'\ia Molina (2013). 'Comparison between internal and external DSLs via RubyTL and Gra2MoL'. In: *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments: Recent Developments*, pp. 109–131. DOI: 10.4018/978-1-4666-2092-6.ch005.

Dalal, Sandeep and Rajender Singh Chhillar (2012). 'Case Studies of Most Common and Severe Types of Software System Failure'. In: *International Journal of Advanced Research in Computer Science and Software Engineering* 2.8, pp. 341–347. ISSN: 2277 128X.

Dantzig, G. B. and J. H. Ramser (1959). 'The Truck Dispatching Problem'. In: *Management Science* 6.1, pp. 80–91. ISSN: 00251909, 15265501. URL: http://www.jstor.org/stable/2627477.

Dantzig, George, Ray Fulkerson and Selmer Johnson (1954). 'Solution of a large-scale traveling-salesman problem'. In: *Journal of the Operations Research Society of America* 2.4, pp. 393–410.

Da Silva, Bruno Castro, Ana L. C. Bazzan, Gustavo K. Andriotti, Filipe Lopes and Denise de Oliveira (2006). 'ITSUMO: An Intelligent Transportation System for Urban Mobility'. In: *Innovative Internet Community Systems: 4th International Workshop, IICS 2004, Guadalajara, Mexico, June 21-23, 2004. Revised Papers*. Ed. by Thomas Böhme, Victor M. Larios Rosillo, Helena Unger and Herwig Unger. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 224–235. DOI: 10.1007/11553762_22.

Davidsson, Paul, Lawrence Henesey, Linda Ramstedt, Johanna Törnquist and Fredrik Wernstedt (2005). 'Agent-Based Approaches to Transport Logistics'. In: *Applications of Agent Technology in Traffic and Transportation*. Ed. by Franziska Klügl, Ana Bazzan and Sascha Ossowski. Basel: Birkhäuser Basel, pp. 1–15. DOI: 10.1007/3-7643-7363-6_1.

Dennett, D. C. (1987). *The Intentional Stance*. A Bradford book. MIT Press. ISBN: 9780262040938. URL: https://books.google.de/books?id=MXbenQEACAAJ.

De Sousa, Luís Moreira and Alberto Rodrigues da Silva (2018). 'Usability evaluation of the domain specific language for spatial simulation scenarios'. In: *Cogent Engineering* 5.1. Ed. by Stefania Tomasiello, p. 1436889. DOI: 10.1080/23311916.2018.1436889.

Do Nascimento, Leandro Marques, Daniel Leite Viana, Paulo Silveira Am Neto, Dhiego A. O. Martins, Vinicius Cardoso Garcia and Silvio R. L. Meira (2012). 'A systematic mapping study on domain-specific languages'. In: *Proceedings of the 7th International Conference on Software Engineering Advances (ICSEA'12)*, pp. 179–187.

Dorigo, M. and L. M. Gambardella (Apr. 1997). 'Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem'. In: *IEEE Transactions on Evolutionary Computation* 1.1, pp. 53–66. DOI: 10.1109/4235.585892.

Dwarakanath, Anurag, Dipin Era, Aditya Priyadarshi, Neville Dubash and Sanjay Podder (2017). 'Accelerating Test Automation through a Domain Specific Language'. In: *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pp. 460–467. DOI: 10.1109/ICST.2017.52.

Ewais, Ahmad and Olga de Troyer (2014). 'A Usability Evaluation of Graphical Modelling Languages for Authoring Adaptive 3D Virtual Learning Environments'. In: *Proceedings of the 6th International Conference on Computer Supported Education - Volume 1*. CSEDU 2014. Setubal, PRT: SCITEPRESS - Science and Technology Publications, Lda, pp. 459–466. DOI: 10.5220/0004947204590466.

Farrenkopf, Thomas, Michael Guckert, Benjamin Hoffmann and Neil Urquhart (2014). 'AGADE'. In: *Multiagent System Technologies*. Ed. by JörgP. Müller, Michael Weyrich and AnaL.C Bazzan. Vol. 8732. Lecture Notes in Computer Science. Springer International Publishing, pp. 234–250. DOI: 10.1007/978-3-319-11584-9_16.

Fellendorf, Martin and Peter Vortisch (2010). 'Microscopic Traffic Flow Simulator VISSIM'. In: *Fundamentals of Traffic Simulation*. Ed. by Jaume Barceló. Vol. 145. International Series in Operations Research & Management Science. New York, NY: Springer New York, pp. 63–93. DOI: 10.1007/978-1-4419-6142-6_2.

Flood, Merrill M. (1956). 'The Traveling-Salesman Problem'. In: *Operations Research* 4.1, pp. 61–75. URL: http://www.jstor.org/stable/167517.

ForterTeam (2019). *Shopping Cart Second Thoughts: A new study of 2,000 Americans found...* Ed. by Forter. URL: https://www.forter.com/blog/infographic-customers-wont-tolerate-friction-filled-checkout/ (visited on 22/09/2021).

Fowler, Martin and Rebecca Parsons (2011). *Domain-specific languages*. A Martin Fowler signature book. Upper Saddle River, NJ, Boston and Indianapolis: Addison-Wesley. ISBN: 9780321712943.

Al-Furhud, Maha Ata and Zakir Hussain Ahmed (2020). 'Genetic algorithms for the multiple travelling salesman problem'. In: *International Journal of Advanced Computer Science and Applications (IJACSA)* 11.7, pp. 553–560. DOI: 10.14569/IJACSA.2020.0110768.

Gabriel, Pedro, Miguel Goulão and Vasco Amaral (2010). 'Do Software Languages Engineers Evaluate their Languages?' In: *Proceedings of the XIII Congreso Iberoamericano en "Software Engineering" (CIbSE'2010)* abs/1109.6794. DOI: 10.48550/arXiv.1109.6794.

Ghosh, Debasish (2011a). 'DSL for the uninitiated'. In: *Communications of the ACM* 54.7, pp. 44–50. DOI: 10.1145/1965724.1965740.

Ghosh, Debasish (2011b). *DSLs in action*. Greenwich, Conn: Manning. ISBN: 9781935182450.

Hahn, Christian (2008). 'A Domain Specific Modeling Language for Multiagent Systems'. In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*. AAMAS '08. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, pp. 233–240. DOI: 10.5555/1402383.1402420.

Hausberger, Stefan (2003). *Simulation of real world vehicle exhaust emissions: Zugl.: Graz, Univ., Habil.-Schr., 2003*. Vol. 82. VKM-THD Mitteilungen. Graz: Verl. der Techn. Univ. ISBN: 3901351744.

Hermans, Felienne, Martin Pinzger and Arie van Deursen (2009). 'Domain-Specific Languages in Practice: A User Study on the Success Factors'. In: *Model Driven Engineering Languages and Systems*. Ed. by Andy Schürr and Bran Selic. Berlin,

Heidelberg: Springer Berlin Heidelberg, pp. 423–437. DOI: 10.1007/978-3-642-04425-0_33.

Hevner, A. R., S. T. March, J. Park and S. Ram (2004). 'Design Science in Information Systems Research'. In: *MIS Quarterly* 28.1, pp. 75–105. ISSN: 02767783.

Hevner, Alan R., Richard C. Linger, Rosann W. Collins, Mark G. Pleszkoch, Stacy J. Prowell and Walton Gwendolyn H. (2005). *The impact of function extraction technology on next-generation software engineering*. Tech. rep. CMU/SEI-2005-TR-015. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. DOI: 10.1184/R1/6585140.v1.

Hirschmann, Karin, Michael Zallinger, Martin Fellendorf and Stefan Hausberger (2010). 'A new method to calculate emissions with simulated traffic conditions'. In: *13th International IEEE Conference on Intelligent Transportation Systems*, pp. 33–38. DOI: 10.1109/ITSC.2010.5625030.

Hoffmann, B., M. Guckert, T. Farrenkopf, K. Chalmers and N. Urquhart (2018a). 'A Domain-Specific Language For Routing Problems'. In: ed. by Lars Nolle, Alexandra Burger, Jens Werner Christoph Tholen and Jens Wellhausen. 32nd European Conference on Modelling and Simulation, Wilhelmshaven, Germany, May 22nd – May 265h, 2018. European Council for Modeling and Simulation, pp. 262–268. DOI: 10.7148/2018-0262.

Hoffmann, Benjamin, Kevin Chalmers, Neil Urquhart, Thomas Farrenkopf and Michael Guckert (2018b). 'Towards Reducing Complexity of Multi-agent Simulations by Applying Model-Driven Techniques'. In: *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection*. Ed. by Yves Demazeau, Bo An, Javier Bajo and Antonio Fernández-Caballero. Cham: Springer International Publishing, pp. 187–199. DOI: 10.1007/978-3-319-94580-4_15.

Hoffmann, Benjamin, Kevin Chalmers, Neil Urquhart and Michael Guckert (2019a). 'Athos - A Model Driven Approach to Describe and Solve Optimisation Problems: An Application to the Vehicle Routing Problem with Time Windows'. In: *Proceedings of the 4th ACM International Workshop on Real World Domain*

Specific Languages. RWDSL '19. New York, NY, USA: ACM, pp. 1–10. DOI: 10.1145/3300111.3300114.

Hoffmann, Benjamin, Michael Guckert, Kevin Chalmers and Neil Urquhart (2019b). 'Simulating Dynamic Vehicle Routing Problems With Athos'. In: *ECMS 2019 Proceedings edited by Mauro Iacono, Francesco Palmieri, Marco Gribaudo, Massimo Ficco*. ECMS, pp. 296–302. DOI: 10.7148/2019-0296.

Hoffmann, Benjamin, Neil Urquhart, Kevin Chalmers and Michael Guckert (2020). 'Athos: An Extensible DSL for Model Driven Traffic and Transport Simulation'. In: *Modellierung 2020*. Ed. by Dominik Bork, Dimitris Karagiannis and Heinrich C. Mayr. Bonn: Gesellschaft für Informatik e.V, pp. 141–156.

Hoffmann, Benjamin, Neil Urquhart, Kevin Chalmers and Michael Guckert (2022). 'An empirical evaluation of a novel domain-specific language – modelling vehicle routing problems with Athos'. In: *Empirical software engineering* 27.7, p. 180. DOI: 10.1007/s10664-022-10210-w.

Holland, John H. (1992). 'Complex adaptive systems'. In: *Daedalus* 121.1, pp. 17–30. URL: http://www.jstor.org/stable/20025416.

Ingibergsson, Johann Thor Mogensen, Stefan Hanenberg, Joshua Sunshine and Ulrik Pagh Schultz (2018). 'Experience Report: Studying the Readability of a Domain Specific Language'. In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. SAC '18. New York, NY, USA: Association for Computing Machinery, pp. 2030–2033. DOI: 10.1145/3167132.3167436.

Iung, Aníbal, João Carbonell, Luciano Marchezan, Elder Rodrigues, Maicon Bernardino, Fabio Paulo Basso and Bruno Medeiros (2020). 'Systematic mapping study on domain-specific language development tools'. In: *Empirical Software Engineering* 25.5, pp. 4205–4249. DOI: 10.1007/s10664-020-09872-1.

Johanson, Arne N. and Wilhelm Hasselbring (2017). 'Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment'. In: *Empirical Software Engineering* 22.4, pp. 2206–2236. DOI: 10.1007/s10664-016-9483-z.

Johnsson, Mikael and Alexancer Olsson (2016). 'Xtext language-based editor'. In: Project in Computer Science - EDAN70. URL: https://fileadmin.cs.lth.se/cs/Education/edan70/CompilerProjects/2015/Reports/JohnssonOlsson.pdf.

Junjie, Pan and Wang Dingwei (2006). 'An ant colony optimization algorithm for multiple travelling salesman problem'. In: *First International Conference on Innovative Computing, Information and Control-Volume I (ICICIC'06)*. Vol. 1, pp. 210–213. DOI: 10.1109/ICICIC.2006.40.

Kahraman, Gökhan and Semih Bilgen (2015). 'A framework for qualitative assessment of domain-specific languages'. In: *Software & Systems Modeling* 14.4, pp. 1505–1526. DOI: 10.1007/s10270-013-0387-8.

Kardas, Geylani, Baris Tekin Tezel and Moharram Challenger (2018). 'Domain–specific modelling language for belief–desire–intention software agents'. In: *IET Software* 12.4, pp. 356–364. DOI: 10.1049/iet-sen.2017.0094.

Kardoš, Martin and Matilda Drozdová (2010). 'Analytical method of CIM to PIM transformation in Model Driven Architecture (MDA)'. In: *Journal of information and organizational sciences* 34.1, pp. 89–99.

Karsai, Gabor, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler and Steven Völkel (2014). 'Design Guidelines for Domain Specific Languages'. In: *ArXiv* abs/1409.2378, pp. 7–13.

Korte, B. (2008). 'The Traveling Salesman Problem'. In: *Combinatorial Optimization: Theory and Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 527–562. DOI: 10.1007/978-3-540-71844-4_21.

Kosar, Tomaž, Sudev Bohra and Marjan Mernik (2016). 'Domain-Specific Languages: A Systematic Mapping Study'. In: *Information and Software Technology* 71, pp. 77–91. DOI: 10.1016/j.infsof.2015.11.001.

Kosar, Tomaž, Sašo Gaberc, Jeffrey C. Carver and Marjan Mernik (2018). 'Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments'. In: *Empirical Software Engineering* 23.5, pp. 2734–2763. DOI: 10.1007/s10664-017-9593-2.

Kosar, Tomaž, Marjan Mernik and Jeffrey C. Carver (2012). 'Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments'. In: *Empirical Software Engineering* 17.3, pp. 276–304. DOI: 10.1007/s10664-011-9172-x.

Kosar, Tomaž, Nuno Oliveira, Marjan Mernik, Maria João Pereira, Matej Crepinsek, Daniela Cruz and Pedro Henriques (2010). 'Comparing general-purpose and domain-specific languages: An empirical study'. In: *ComSIS–Computer Science an Information Systems Journal* 7.2, pp. 247–264. DOI: 10.2298/CSIS1002247K.

Krajzewicz, Daniel, Jakob Erdmann, Michael Behrisch and Laura Bieker (2012). 'Recent development and applications of SUMO-Simulation of Urban MObility'. In: *International Journal on Advances in Systems and Measurements* 5.3, pp. 128–138.

Laporte, Gilbert and Yves Nobert (1987). 'Exact algorithms for the vehicle routing problem'. In: *North-Holland Mathematics Studies*. North-Holland Mathematics Studies 132. Ed. by Silvano Martello, Gilbert Laporte, Michel Minoux and Celso Ribeiro, pp. 147–184. DOI: 10.1016/S0304-0208(08)73235-3.

Larrañaga, P., C.M.H. Kuijpers, R. H. Murga, I. Inza and S. Dizdarevic (1999). 'Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators'. In: *Artificial Intelligence Review* 13.2, pp. 129–170. DOI: 10.1023/A:1006529012972.

Lim, Stanley Frederick W.T., Xin Jin and Jagjit Singh Srai (2018). 'Consumer-driven e-commerce'. In: *International Journal of Physical Distribution & Logistics Management* 48.3, pp. 308–332. DOI: 10.1108/IJPDLM-02-2017-0081.

Lind, Jürgen (2001). 'Issues in agent-oriented software engineering'. In: *AOSE 2000. Lecture Notes in Computer Science*. Ed. by Paolo Ciancarini and Michael J. Wooldridge. Vol. 1957. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 45–58. DOI: 10.1007/3-540-44564-1_3.

Lu, Ruqian and Zhi Jin (2000). 'Domain modeling-based software engineering: a formal approach'. In: vol. 8. Springer Science & Business Media, p. 123. ISBN: 9781461544876.

Macal, C. M. (2016). 'Everything you need to know about agent-based modelling and simulation'. In: *Journal of Simulation* 10.2, pp. 144–156. DOI: 10.1057/jos.2016.7.

Macal, C. M. and M. J. North (2010). 'Tutorial on agent-based modelling and simulation'. In: *Journal of Simulation* 4.3, pp. 151–162. DOI: 10.1057/jos.2010.3.

Maidstone, Robert (2012). 'Discrete event simulation, system dynamics and agent based simulation: Discussion and comparison'. In: *System* 1.6, pp. 1–6.

McKnight, Patrick E. and Julius Najab (2010). 'Mann-Whitney U Test'. In: *The Corsini Encyclopedia of Psychology*. Ed. by Irving B. Weiner and W. Edward Craighead. Vol. 18. Hoboken, NJ, USA: John Wiley & Sons, Inc, p. 50. DOI: 10.1002/9780470479216.corpsy0524.

Meliá, Santiago, Cristina Cachero, Jesús M. Hermida and Enrique Aparicio (2016). 'Comparison of a textual versus a graphical notation for the maintainability of MDE domain models: an empirical pilot study'. In: *Software Quality Journal* 24.3, pp. 709–735. DOI: 10.1007/s11219-015-9299-x.

Meliá, Santiago, Jaime Gómez, Sandy Pérez and Oscar Díaz (2008). 'A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA'. In: *2008 Eighth International Conference on Web Engineering*, pp. 13–23. DOI: 10.1109/ICWE.2008.36.

Mernik, Marjan, Jan Heering and Anthony M. Sloane (2005). 'When and How to Develop Domain-specific Languages'. In: *ACM Comput. Surv.* 37.4, pp. 316–344. DOI: 10.1145/1118890.1118892.

Miller, Clair E., Albert W. Tucker and Richard A. Zemlin (1960). 'Integer programming formulation of traveling salesman problems'. In: *Journal of the ACM (JACM)* 7.4, pp. 326–329. DOI: 10.1145/321043.321046.

Mock, Kenrick J, J. Ward Testa, Cameron Taylor, Heather Koyuk, Jessica Coyle, Russell Waggoner and Kelly Newman (2007). 'Final Report: An Agent-Based Model of Predator-Prey Relationships Between Transient Killer Whales and Other Marine Mammals: University of Alaska Anchorage, Anchorage'. In: URL: http://www.

cse.uaa.alaska.edu/~urps/files/136/Mock_Testa_MMCFinalReport.pdf (visited on 10/06/2017).

North, Michael J. and Charles M. Macal (2009). 'Agent Based Modeling and Computer Languages'. In: *Encyclopedia of Complexity and Systems Science*. Ed. by Robert A. Meyers. New York, NY: Springer New York, pp. 131–148. DOI: 10.1007/978-0-387-30440-3_8.

North, Michael J. *et al.* (2010). 'Multiscale agent-based consumer market modeling'. In: *Complexity* 15.5, pp. 37–47. DOI: 10.1002/cplx.20304.

Ombuki, Beatrice, Brian J. Ross and Franklin Hanshar (2006). 'Multi-Objective Genetic Algorithms for Vehicle Routing Problem with Time Windows'. In: *Applied Intelligence* 24.1, pp. 17–30. DOI: 10.1007/s10489-006-6926-z.

OMG (2014). *Object Constraint Language: Version 2.4*. Ed. by Object Management Group. URL: http://www.omg.org/spec/OCL/2.4.

OMG (2017). *OMG Unified Modeling Language*. URL: https://www.omg.org/spec/UML/2.5.1/PDF.

Orman, A. J. and H. P. Williams (2007). 'A Survey of Different Integer Programming Formulations of the Travelling Salesman Problem'. In: *Optimisation, Econometric and Financial Analysis*. Ed. by Erricos John Kontoghiorghes and Cristian Gatu. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 91–104. DOI: 10.1007/3-540-36626-1_5.

Padgham, Lin and Michael Winikoff (2005). 'Prometheus'. In: *Agent-Oriented Methodologies*. Ed. by Brian Henderson-Sellers and Paolo Giorgini. IGI Global, pp. 107–135. DOI: 10.4018/978-1-59140-581-8.ch005.

Palmer, R. G., W. Brian Arthur, John H. Holland and Blake LeBaron (1999). 'An artificial stock market'. In: *Artificial Life and Robotics* 3.1, pp. 27–31. DOI: 10.1007/BF02481484.

Pan, Xiaoshan, Charles S. Han, Ken Dauber and Kincho H. Law (2007). 'A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations'. In: *AI & SOCIETY* 22.2, pp. 113–132. DOI: 10.1007/s00146-007-0126-1.

Papadimitriou, Christos H. and Kenneth Steiglitz (2013). *Combinatorial Optimization: Algorithms and Complexity*. Dover Books on Computer Science. Newburyport: Dover Publications. ISBN: 9780486402581.

Parr, Terence (2011). *Language implementation patterns: Create your own domain-specific and general programming languages*. P3.0 printing, Version: 2011-7-13. The pragmatic programmers. Raleigh, NC: The Pragmatic Bookshelf. ISBN: 9781934356456.

Parry, Hazel R. (2009). 'Agent Based Modeling, Large Scale Simulations'. In: *Encyclopedia of Complexity and Systems Science*. Ed. by Robert A. Meyers. New York, NY: Springer New York, pp. 148–160. DOI: 10.1007/978-0-387-30440-3_9.

Philip Welch (2017). *Developing a commercial dynamic vehicle routing system - a case study*. DOI: 10.13140/RG.2.2.14915.30247.

Pillac, Victor, Michel Gendreau, Christelle Guéret and Andrés L. Medaglia (2013). 'A review of dynamic vehicle routing problems'. In: *European Journal of Operational Research* 225.1, pp. 1–11. DOI: 10.1016/j.ejor.2012.08.015.

Pokahr, Alexander, Lars Braubach and Kai Jander (2013). 'The Jadex Project: Programming Model'. In: *Multiagent Systems and Applications: Volume 1:Practice and Experience*. Ed. by Maria Ganzha and Lakhmi C. Jain. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 21–53. DOI: 10.1007/978-3-642-33323-1_2.

Poltronieri, Ildevana, Allan Christopher Pedroso, Avelino Francisco Zorzo, Maicon Bernardino and Marcia de Borba Campos (2021). 'Is Usability Evaluation of DSL Still a Trending Topic?' In: *Human-Computer Interaction. Theory, Methods and Tools*. Ed. by Masaaki Kurosu. Vol. 12762. Cham: Springer International Publishing, pp. 299–317. DOI: 10.1007/978-3-030-78462-1_23.

Poltronieri, Ildevana, Avelino Francisco Zorzo, Maicon Bernardino and Marcia de Borba Campos (2018). 'Usa-DSL: Usability Evaluation Framework for Domain-Specific Languages'. In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. SAC '18. New York, NY, USA: Association for Computing Machinery, pp. 2013–2021. DOI: 10.1145/3167132.3167348.

Poltronieri Rodrigues, Ildevana, Márcia de Borba Campos and Avelino F. Zorzo (2017). 'Usability Evaluation of Domain-Specific Languages: A Systematic Literature Review'. In: *Human-Computer Interaction. User Interface Design, Development and Multimodality*. Ed. by Masaaki Kurosu. Cham: Springer International Publishing, pp. 522–534. DOI: 10.1007/978-3-319-58071-5_39.

Quak, Hans and Bram Kin (2020). 'The impact of future delivery models in last-mile home deliveries'. In: *Green Cities 2020 – 4th International Conference – Green Logistics for Greener Cities*. Szczecin, Poland.

Rao, Anand S. and Michael P. Georgeff (1991). 'Modeling rational agents within a BDI-architecture'. In: KR'91 91, pp. 473–484.

Sansores, Candelaria and Juan Pavón (2005). 'Agent-Based Simulation Replication: A Model Driven Architecture Approach'. In: *MICAI 2005: Advances in Artificial Intelligence*. Ed. by Alexander Gelbukh, Álvaro de Albornoz and Hugo Terashima-Marín. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 244–253. DOI: 10.1007/11579427_25.

Segal, Judith (2009). 'Software Development Cultures and Cooperation Problems: A Field Study of the Early Stages of Development of Software for a Scientific Community'. In: *Computer Supported Cooperative Work (CSCW)* 18.5, p. 581. DOI: 10.1007/s10606-009-9096-9.

Seth Tisue and Uri Wilensky (2004). 'Netlogo: A simple environment for modeling complexity'. In: *Proceedings of the Fifth International Conference on Complex Systems ICCS 2004*, pp. 16–21.

Shannon, Robert E. (1998). 'Introduction to the Art and Science of Simulation'. In: *Proceedings of the 30th Conference on Winter Simulation*. WSC '98. Los Alamitos, CA, USA: IEEE Computer Society Press, pp. 7–14. DOI: 10.1109/WSC.1998.744892.

Sjoeberg, D.I.K., J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg and A. C. Rekdal (2005). 'A survey of controlled experiments in software engineering'. In: *IEEE Transactions on software Engineering* 31.9, pp. 733–753. DOI: 10.1109/TSE.2005.97.

Skylar Ross (March 2021). *Seven Last-Mile Delivery Challenges, and How to Solve Them*. URL: https://www.supplychainbrain.com/blogs/1-think-tank/post/32800-last-mile-delivery-challenges-and-how-to-solve-them (visited on 22/09/2021).

Soti (2020). *The last mile sprint: State of mobility in transportation and logistics*. Ed. by Soti Inc. URL: https://soti.net/lp/state-of-mobility-in-transportation-and-logistics/ (visited on 23/09/2021).

Stahl, Thomas (2007). *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. 2., aktualisierte und erw. Aufl. Heidelberg: Dpunkt-Verl. ISBN: 9783898644488.

Statista (2021). *Biggest challenges for logistics providers in last mile delivery in the United States in 2020 [Graph]*. Ed. by Statista. URL: https://www.statista.com/statistics/816884/last-mile-delivery-logistics-providers-challenges/ (visited on 14/09/2021).

Steinberg, Dave (2009). *EMF: Eclipse Modeling Framework*. 2nd ed., Rev. and updated. The eclipse series. Upper Saddle River, NJ: Addison-Wesley. ISBN: 9780321331885.

Sterman, John D. (2000). *Business dynamics: Systems thinking and modeling for a complex world*. [Nachdr.] Boston: Irwin/McGraw-Hill. ISBN: 007238915X.

Taillandier, Patrick, Benoit Gaudou, Arnaud Grignard, Quang-Nghi Huynh, Nicolas Marilleau, Philippe Caillou, Damien Philippon and Alexis Drogoul (2019a). 'Building, composing and experimenting complex spatial models with the GAMA platform'. In: *GeoInformatica* 23.2, pp. 299–322. DOI: 10.1007/s10707-018-00339-6.

Taillandier, Patrick, Arnaud Grignard, Nicolas Marilleau, Damien Philippon, Quang-Nghi Huynh, Benoit Gaudou and Alexis Drogoul (2019b). 'Participatory Modeling and Simulation with the GAMA Platform'. In: *Journal of Artificial Societies and Social Simulation* 22.2, p. 3. DOI: 10.18564/jasss.3964.

Thorne, David R. (2006). 'Throughput: a simple performance index with desirable characteristics'. In: *Behavior Research Methods* 38.4, pp. 569–573. DOI: 10.3758/BF03193886.

Toth, Paolo and Daniele Vigo (2002). 'Models, relaxations and exact approaches for the capacitated vehicle routing problem'. In: *Discrete Applied Mathematics* 123.1, pp. 487–512. DOI: 10.1016/S0166-218X(01)00351-1.

van Deursen, Arie (1997). 'Domain-specific languages versus object-oriented frameworks: A financial engineering case study'. In: *Smalltalk and Java in Industry and Academia, STJA'97*, pp. 35–39.

van Deursen, Arie, Paul Klint, Joost Visser *et al.* (2000). 'Domain-specific languages: An annotated bibliography'. In: *ACM Sigplan Notices* 35.6, pp. 26–36.

Vandierendonck, André (2017). 'A comparison of methods to combine speed and accuracy measures of performance: A rejoinder on the binning procedure'. In: *Behavior Research Methods* 49.2, pp. 653–673. DOI: 10.3758/s13428-016-0721-5.

Vandierendonck, André (2018). 'Further Tests of the Utility of Integrated Speed-Accuracy Measures in Task Switching'. In: *Journal of cognition* 1.1, p. 8. DOI: 10.5334/joc.6.

Vangheluwe, Hans, Ximeng Sun and Eric Bodden (2007). 'Domain-Specific Modelling With Atom3'. In: *ICSOFT (PL/DPS/KE/MUSE)*, pp. 298–304.

Vendrov, Ivan, Christopher Dutchyn and Nathaniel D. Osgood (2014). 'Frabjous: A Declarative Domain-Specific Language for Agent-Based Modeling'. In: *Social Computing, Behavioral-Cultural Modeling, and Prediction*. Ed. by William G. Kennedy, Nitin Agarwal and Shanchieh Jay Yang. Vol. 8393. Lecture notes in computer science Information systems and application, incl. Internet/web and HCI. Berlin: Springer International Publishing, pp. 385–392. DOI: 10.1007/978-3-319-05579-4_47.

Visser, Eelco (2008). 'WebDSL: A case study in domain-specific language engineering'. In: *Generative and Transformational Techniques in Software Engineering II*. Ed.

by Ralf Lämmel, Joost Visser and João Saraiva. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 291–373. DOI: 10.1007/978-3-540-88643-3_7.

Völter, Markus and Sebastian Benz (2013). *DSL engineering: Designing, implementing and using domain-specific languages*. [S.l.]: CreateSpace Independent Publishing Platform. ISBN: 9781481218580.

WEF (2020). *The Future of the Last-Mile Ecosystem: Transition Roadmaps for Public- and Private-Sector Players*. Ed. by World Economic Forum. Report. URL: http://www3.weforum.org/docs/WEF_Future_of_the_last_mile_ecosystem.pdf (visited on 19/09/2021).

WEF (2021). *Pandemic, Parcels and Public Vaccination: Envisioning the NExt Normal for the Last-Mile Ecosystem*. Ed. by World Economic Forum. URL: http://www3.weforum.org/docs/WEF_Pandemic_Parcels_and_Public_Vaccination_report_2021.pdf (visited on 20/09/2021).

Winikoff, Michael (2005). 'Jack™ Intelligent Agents: An Industrial Strength Platform'. In: *Multi-Agent Programming: Languages, Platforms and Applications*. Ed. by Rafael H. Bordini, Mehdi Dastani, Jürgen Dix and Amal El Fallah Seghrouchni. Boston, MA: Springer US, pp. 175–193. DOI: 10.1007/0-387-26350-0_7.

Woltz, Dan J. and Christopher A. Was (2006). 'Availability of related long-term memory during and after attention focus in working memory'. In: *Memory & Cognition* 34.3, pp. 668–684. DOI: 10.3758/BF03193587.

Wooldridge, Michael and Nicholas R. Jennings (1995). 'Intelligent agents: Theory and practice.' In: *The Knowledge Engineering Review* 10.2, pp. 115–152. DOI: 10.1017/S0269888900008122.

Woolson, R. F. (2007). 'Wilcoxon Signed-Rank Test'. In: *Wiley Encyclopedia of Clinical Trials*. Ed. by Ralph B. D'Agostino, Lisa Sullivan and Joseph Massaro. Vol. 62. Hoboken, NJ, USA: John Wiley & Sons, Inc. DOI: 10.1002/9780471462422.eoct979.

Yoav Shoham (1993). 'Agent-oriented programming'. In: *Artificial Intelligence* 60.1, pp. 51–92. DOI: 10.1016/0004-3702(93)90034-9.

Appendix A

The Athos syntax

A.1 The Xtext grammar definition

```
grammar de.thm.mnd.wbm.Athos5 with org.eclipse.xtext.common.Terminals

import "http://www.eclipse.org/emf/2002/Ecore" as ecore
import "http://www.thm.de/mnd/wbm/graph" as graph

generate athos5 "http://www.thm.de/mnd/wbm/Athos5"

Model:
  'model' name=ID
  (world = World)?
  (productDefinitions=ProductDefinitions)?
  functions = Functions
  network = Network
  (agentAttributes = AgentAttributes)?
  (agentTypes = AgentTypes)?
  (metricSection = MetricSection)? ;

World:
     '['{World} ('xmin' xmin=Double)?  ('xmax' xmax=Double)?
          ('ymin' ymin=Double)?  ('ymax' ymax=Double)? ']' ;

Functions:
  'functions' {Functions}
  functions += Function (','? functions+=Function)* ;

Function returns Function:
  EdgeFunction | AgentFunction ;

EdgeFunction:
  {DurationFunction} 'durationFunction' name=ID
      expression=Expression (default?='default')? |
  {SpeedFunction} 'speedFunction' name=ID
      expression=Expression (default?='default')? ;

AgentFunction:'agentFunction' name=ID expression=Expression ;
```

```
ProductDefinitions:
  'products' products+=Product (','? products+=Product)* ;

Product: name=ID ('weight'weight=Double)?
    ('volume' volume=Double)? ('profit' profit=Double)? ;

AgentAttributes:
  'agentAttributes' agentAttributes += AgentAttribute
      (','? agentAttributes += AgentAttribute)* ;

AgentAttribute:
  name=ID ;

AgentTypes:
  'agentTypes' agentTypes += AgentType+ ;

AgentType:
  {AgentType} 'agentType' name=ID ('congestionFactor' congestionFactor=Double)?
      ('maxVolume' maxVolume=Double)?  ('maxWeight' maxWeight=Double)?
      (attributeAssigments += AttributeAssignment)*
      behaviour = AgentBehaviourBlock
      (individualOptimization ?= 'optimises' function=[AgentFunction])? ;

AgentBehaviourBlock:
  {AgentBehaviourBlock}  (agentBehaviourStates+=AgentBehaviourState)+ ;


AgentBehaviourState:
  'behaviour' name=ID description=AgentBehaviourDescription
      (transition+=AgentBehaviourTransition)* ;

AgentBehaviourTransition:
  'when' (condition=Expression) 'do' refState=[AgentBehaviourState]
      (resume?='resume')? ;

AgentBehaviourDescription:
  AgentLoadingBehaviour |
  AgentIdlingBehaviour |
  AgentStaticDestinationBehaviour |
  AgentReturnToDepotBehaviour |
  AgentResumeStaticDeliveryBehaviour |
  AgentVanishingBehaviour |
  AgentAwaitTourFromDepotBehaviour |
  AgentAwaitTourFromDepotExternalBehaviour|
  NodeExtendedOptimisationBehaviour|
  AgentRoutingBehaviour ;

AgentRoutingBehaviour:
  AgentStrictDeliveryBehaviour |
  AgentExactTourFollowingBehaviour  ;

NodeTourOptimisationBehaviour:
  NodeStaticTourOptimisationBehaviour |
  NodeStaticTourAntOptimisationBehaviour |
  NodeStaticTourEAOptimisationBehaviour ;
```

```
// BEGIN: Experimental feature: Extension mechanism via algorihm extensions

AgentExtendedOptimisationBehaviour:
  'extended'  name=ID ((keys+=ID) (values+=ValueList))* ;

ValueList:
  StringValue | StringList | StringListList | DoubleValue | DoubleList |
      DoubleListList | (=>IntegerValue) | (=>IntegerList)|(=>IntegerListList) |
      NodeValue | NodeList | NodeListList ;

StringValue:
  stringValue=STRING ;

StringList:
  '('stringList+=STRING (',' stringList+=STRING)* ')' ;

StringListList:
  '('outerStringList+=StringList (',' outerStringList+=StringList)*')' ;

DoubleValue:
  doubleValue=Double ;

DoubleList:
  '(' doubleList+=Double (',' doubleList+=Double)* ')' ;

DoubleListList:
  '(' outerDoubleList+=DoubleList (',' outerDoubleList+=DoubleList)* ')' ;

IntegerValue:
  integerList=INT ;

IntegerList:
  '(' integerList+=INT (',' integerList+=INT)* ')' ;

IntegerListList:
  '(' outerIntegerList+=IntegerList (',' outerIntegerList+=IntegerList)* ')' ;

// END: Experimental feature: Extension mechanism via algorihm extensions

NodeValue:
  nodeValue=[graph::Node] ;

NodeList:
  '(' nodeList+=[graph::Node] (',' nodeList+=[graph::Node])* ')' ;

NodeListList:
  '(' outerNodeList+=NodeList (',' outerNodeList+=NodeList)* ')' ;

AgentVanishingBehaviour:
  {AgentVanishingBehaviour} 'vanish' ;

AgentAwaitTourFromDepotBehaviour:
  {AgentAwaitTourFromDepotBehaviour} 'awaitTour' ;

AgentAwaitTourFromDepotExternalBehaviour:
  {AgentAwaitTourFromDepotExternalBehaviour} 'awaitTourExternal' ;

AgentResumeStaticDeliveryBehaviour:
  AgentResumeStaticDeliveryBehaviourAtLast |
    AgentResumeStaticDeliveryBehaviourAtNext ;
```

```
AgentResumeStaticDeliveryBehaviourAtLast:
  'resume' targetState=[AgentBehaviourState] 'at' 'last' ;

AgentResumeStaticDeliveryBehaviourAtNext:
  'resume' targetState=[AgentBehaviourState] 'at' 'next' ;

AgentIdlingBehaviour:
  'idle' 'for' idleTime=Expression ;

AgentReturnToDepotBehaviour:
  'returnToDepot' (nearest?='nearest'|node=[graph::Node]) ;

AgentLoadingBehaviour:
  'loadCargo' cargo+=ProductAndQuantity (',' cargo+=ProductAndQuantity)* ;

AgentStrictDeliveryBehaviour:
  'deliver'  route+=[graph::Node] (','? route+=[graph::Node])*
      (everything?='everything' |
  'products' products+=[Product] (','? products+=[Product])*) ;

AgentExactTourFollowingBehaviour:
  'route'   route+=[graph::Node] (',' route+=[graph::Node])*
    (repeat ?= 'repeat' repetitions=INT 'times')? ;

NodeStaticTourOptimisationBehaviour:
  'customers'   route+=[graph::Node] (','? route+=[graph::Node])* ;

NodeStaticTourEAOptimisationBehaviour:
  'customers' 'ea'  route+=[graph::Node] (','? route+=[graph::Node])*
    (('popSize' popsize=INT)? & ('simplePermuProb' simplePermuProb=Double)? &
      ('maxDistance' maxDistance=Double)? &
      ('generations' generations=INT)? &
      ('weightNoOfTours' weightNoOfTours=Double)? &
      ('weightTotalDistance' weightTotalDistance=Double)? &
      ('tournamentSize' tournamentSize=INT)? &
      ('takeBestProb' takeBestProb=Double)? &
      ('mutationProb' mutationProb=Double)?) ;

NodeStaticTourAntOptimisationBehaviour:
  'customers' 'ant' '(' route+=[graph::Node]
    (',' route+=[graph::Node])* ')'
    ('repeat' repetitions=INT 'times')
    (('ants' numberOfAnts=INT)? &
    ('t0' t0=Double)? &
    ('alpha' alpha=Double)? &
    ('q0' q0=Double)? &
    ('beta' beta=Double)? &
    ('rho' rho=Double)? &
    ('iterations' iterations=INT)?) ;

AgentStaticDestinationBehaviour:
  'destination' node=[graph::Node] ;

AttributeAssignment:
  'attr' attribute = [AgentAttribute] value = Double ;
```

```
Network:
  ({Network}(complete?= 'complete')? 'network'
    ('edgeAttributes' edgeAttributes+=EdgeAttribute
        (','? edgeAttributes+=EdgeAttribute)*)?
    'nodes' nodes += Node (','? nodes+=Node)*
    ('edges' (edges+=Edge (','? edges+=Edge)*)?
        (edgeGroups+=EdgeGroup (','? edgeGroups+=EdgeGroup)*)?)?
    ('sources' sources+=Source (','? sources+=Source)*)?
    ('demands' demands+=Demand (','? demands+=Demand)*)?
  ) ;

EdgeAttribute: name=ID;

EdgeAttributeAssignment:
  'attr' attribute=[EdgeAttribute] value=Double ;

Node returns graph::Node:
  SimpleNode | DemandNode | SourceNode ;

SimpleNode:
   Nodish ;

DemandNode:
  Nodish
  'hasDemand' demandDefinition+=ProductAndQuantityAbsolute
  (','? demandDefinition+=ProductAndQuantityAbsolute)*
  (
    ('timeWindow' ealiestTime=INT (','? latestTime=INT)?) |
    ('earliestTime' ealiestTime=INT) ('latestTime' latestTime=INT)?
  )?
  ('serviceTime' serviceTime=INT)? ;

SourceNode:
  Nodish Sourcish ;

fragment Nodish * : name=ID  'at'? '('? x=Double ','? y=Double ')'?;

fragment Sourcish :
  (isDepot?='isDepot' products+=[Product] (','? products+=[Product])*)?
  ('sprouts' ("quantity" factor=INT)? sproutFunction=SproutFunction)
  (
    ('frequency' frequency=Double ('every' every=INT)? ('until' until=INT)?) |
    (tourOptimisation=NodeTourOptimisationBehaviour ('at' at=INT)?
    ('maxAgents'maxAgents=INT)?) ('latestTime' latestTime=INT)? |
    (simpleStart?='at' at=INT)
  ) ;

Edge returns graph::Edge:
  {graph::Edge} 'vintage' name=ID
  (
    'from' from=[graph::Node] 'to' to=[graph::Node] |
    '('from=[graph::Node] ',' to=[graph::Node] ')'
  ) |
  {FunctionalEdge} (directed?=('directed'|'->') |
  undirected?=('undirected'|'<->'))? (name=ID)?
  (
    'from' from=[graph::Node] 'to' to=[graph::Node] |
    '('from=[graph::Node] ',' to=[graph::Node] ')'
  )
```

```
  (
    edgeAttributeAssignments+=EdgeAttributeAssignment
    (','? edgeAttributeAssignments+=EdgeAttributeAssignment)*
  )?
  (lenExp ?= 'length' length=Double)? ('cfactor' cfactor=Double)?
  ('path' path=STRING)? ('baseSpeed' baseSpeed=Double)?
  ('function' function=[EdgeFunction])?
  (appearance=AppearanceSpecification)? ;

EdgeGroup:
  'group' (name=ID)
  (directed?=('directed'|'->') | undirected?=('undirected'|'<->'))?
  (edgeAttributeAssignments+=EdgeAttributeAssignment
  (','? edgeAttributeAssignments+=EdgeAttributeAssignment)*)?
  ('cfactor' cfactor=Double)?
  ('path' path=STRING)? ('baseSpeed' baseSpeed=Double)?
  ('function' (function=[EdgeFunction] | containedFunction=EdgeFunction) )?
  'members'  (appearance=AppearanceSpecification)?
  edges+=Edge (   ','? edges+=Edge)* ;

AppearanceSpecification:
  '[' {AppearanceSpecification}
    (
      (
        type= (
          'type1' | 'type2' | 'type3'
        ) (',')?
      )? &
      ( color=(
          'lightRed' | 'red' | 'darkRed' |
          'lightBlue' | 'blue' | 'darkBlue' |
          'lightGreen' | 'green' | 'darkGreen'
        )(',')?
      )? &
      ( thickness=(
          'ultraThin' | 'thin' |
          'normal' |
          'thick' | 'ultraThick'
        )(',')? )?
    )
  ']' ;

ProductAndQuantity:
  ProductAndQuantityAbsolute | ProductAndQuantityRelative;

ProductAndQuantityAbsolute:
  product=[Product]  ('units')  quantity=Double ;

ProductAndQuantityRelative:
  product=[Product]  ('percent') quantity=Double ;

Source: {Source} node=[graph::Node]Sourcish;

// BEGIN: ---- METRICS ----------------------------

MetricSection:
  'metrics' {MetricSection}
    'updateRate' updateRate=INT  agentMetrics+=AgentMetric* ;

AgentMetric:
```

```
  'for' agentType=[AgentType]  metrics+=Metric* ;

Metric:
  (individualOrClass=('individual' | 'class'))
  'metric' name = ID 'when'  condition=Expression
  aggregationKind=('set' | 'add') metricFunction=Expression ;

// END: ----- METRICS ----------------------------

Demand:
  node=[graph::Node] 'hasDemand'
  demandDefinition+=ProductAndQuantityAbsolute
    (','? demandDefinition+=ProductAndQuantityAbsolute)*
  (
    ('timeWindow' ealiestTime=INT (',' latestTime=INT)?) |
    ('earliestTime' ealiestTime=INT) ('latestTime' latestTime=INT)?
  )?
  ('serviceTime' serviceTime=INT)? ;

// later it must be possible to define car types and routes
// separately and combine them!
SproutFunction:
  agentProbabilities+= AgentProb
  ('orr' agentProbabilities += AgentProb)* ;

AgentProb:
  {ReferringAgentProb} agentReference = [AgentType]
    (instanceRoutes ?='route' (setOfInstanceRoutes+=InstanceRoute)+)?
    ('probability' probability=INT)? |
  {EncapsulatingAgentProb}
    agentContainment = AgentType ('probability' probability=INT)? ;

InstanceRoute:
  'for' state=[AgentBehaviourState] (
    '('route+=[graph::Node] (','? route+=[graph::Node])* ')' |
      route+=[graph::Node] (','? route+=[graph::Node])*
  ) ;

Double returns ecore::EDouble:
  ('-')? INT ('.' INT)? ;

exDouble returns ecore::EDouble:
  INT '.' INT ;

Expression:
  Or;

Or returns Expression:
  And ({Or.left=current} 'or' right=And)* ;

And returns Expression:
  Equality ({And.left=current} 'and' right=Equality)* ;

Equality returns Expression:
  Comparison ({Equality.left=current}
  op=('!=' | '==') right=Comparison)* ;

Comparison returns Expression:
  PlusOrMinus ({Comparison.left=current}
  op=('<='|'>='|'>'|'<') right=PlusOrMinus)* ;
```

```
PlusOrMinus returns Expression:
  MulOrDiv ({PlusOrMinus.left = current} op=('+' | '-') right=MulOrDiv)* ;

MulOrDiv returns Expression:
  Power ({MulOrDiv.left=current} op=('*'|'/') right=Power)* ;

Power returns Expression:
  Primary ({Power.base=current} '^' exponent=Primary)? ;

Primary returns Expression:
  {BracketedExpression} '('expression = Expression')' |
  BuiltInFunction ;

BuiltInFunction returns Expression :
  {NormalDistributionFunction} "normallyDistributed"
    "mean" mean=Double "standardDeviation" stdDev=Double |
    Atomic ;

Atomic returns Expression:
  {IntConstant} => value = INT |
  {DoubleConstant} value = exDouble |
  {BuiltInIntValue} value= (
      'number' | 'currentTime'
   ) |
  {BuiltInDoubleValue} value=(
      'accCongestionFactor' |'avgCongestionFactor' |
      'length'| 'cfactor' | 'baseSpeed' | 'accTime'
  ) |
  {AttributeReference} value=[AgentAttribute] |
  {DistanceToExpression} 'distanceTo'
    (fixedExpression?='last'( who=('customer' | 'node')) |node=[graph::Node]) |
  {QuantityOfExpression} 'quantityOf' product=[Product] |
  {IsAtCustomerExpression} 'isAtCustomer?' |
  {EarliestTimeExpression} 'earliestTime' |
  {LatestTimeExpression} 'latestTime' |
  {CreatedNewTourExpression} 'createdNewTour?'|
  {IntendedTourExpression} 'intendedTour' |
  {IsAtNodeExpression} 'isAt?' node = [graph::Node] |
  {MetricRefExpression} 'metric' metric=[Metric] |
  {FinishedStateExpression}'finished' |
  {LastVisitedNode} 'lastVisitedNode' |
  {LastVisitedCustomer} 'lastVisitedCustomer'|
  {NotYetSetExpression} 'notYetSet?' |
  {ColorConstant}value=(
    'red'|'darkred'|'lightred' |
    'blue'|'darkblue'|'lightblue' |
    'green' | 'darkgreen' | 'lightgreen'
  ) ;
```

Appendix B

Evolotionary Algorithm

B.1 Implementation

Listing B.1: Implementation of Ombukis EA.

```
1  /* @author Benjamin Hoffmann (based on the work of Ombuki et al.)
2   *
3   * @param <V>
4   *           Node (Vertex) type used by the external simulation platform.
5   * @param <E>
6   *           Edge (Arc) type used by the external simulation platform.
7   * @param <C>
8   *           Collection type for the storage of the list of used by the external
9   *           simulation platform.
10  *
11  * This class provides an evolutionary algorithm for the vehicle routing problem
12  * with time windows (VRPTW) based on the work of Ombuki et al., 2006.
13  *
14  * Ombuki, B., Ross, B. J. y Hanshar, F. (2006). Multi-Objective Genetic Algorithms
15  * for Vehicle Routing Problem with Time Windows. Applied Intelligence, 24(1), 17-30.
16  * https://doi.org/10.1007/s10489-006-6926-z    */
17 public class VRPTWSolverOmbuki<V, E, C> {
18    /* The VRPTW comprises a directed grap G=(C,A) where C is the set of customers
19     * (Turtles) and A the set of arcs (Links).  */
20    private DirectedSparseMultigraph<V, E> graph2;
21
22    /* Each Turtle is represented by exactly one number and each number is
23     * associated with exactly one turtle.*/
24    private BidiMap<V, Integer> agentIntegerBidiMap2;
25    private HashMap<Integer, Double> integerEarliestTimeMap2;
26    private HashMap<Integer, Double> integerLatestTimeMap2;
27    private HashMap<Integer, Double> integerServiceTimeMap2;
28    private HashMap<Integer, Double> integerDemandMap2;
29
30    /* Each vehicle of the homogeneous fleet has a given capacity which is
31     * represented by this value.*/
32    private double vehicleCapacity;
33
34    /* Each path that connects two agents of the tour is associated with a cost.
35     * This two dimensional array represents the cost matrix c_ij */
36    private double[][] costMatrix2;
37
38    /* Each path that connects two agents of the tour is associated with a time.
39     * This two dimensional array represents the time matrix t_ij */
40    private double[][] timeMatrix2;
41
42    /** Each chromosome is associated with a total length. */
43    private Map<ArrayList<ArrayList<Integer>>, Double> toursTotalLength2;
44
```

```
45    /** The city from which each tour must start */
46    private V startCity2;
47
48    /** The city in which each tour must end */
49    private V endCity2;
50
51
52    /* The initial population of chromosomes - will be transformed into a list of
53     * lists of tours */
54    Integer[][] chromosomes2;
55
56    /* The list of lists of tours the initial population is transformed into and
57     * that evolves throughout the runtime of the algorithm */
58    private ArrayList<ArrayList<ArrayList<Integer>>> population2;
59
60    /** PARETO. Used in the "rankChromosomes()"-method */
61    private Map<ArrayList<ArrayList<Integer>>, Integer> rankOfTours2;
62
63    /* Based on the number of deployed vehicles and the total distance covered, each
64     * list of tours is assigned a weighted sum. In this list, the lists of tours
65     * are sorted according to this sum in an ascending order. */
66    private ArrayList<ArrayList<ArrayList<Integer>>> weightOrderedSumSortedList2;
67
68    /* Based on the number of deployed vehicles and the total distance covered, each
69     * list of tours is assigned a weighted sum. Via this list, it is possible to
70     * obtain this sum for every list of tours. */
71    private Map<ArrayList<ArrayList<Integer>>, Double> weightedSumSortedMap;
72
73    /** Set in method 'generateMatingPopulation()' */
74    private ArrayList<ArrayList<ArrayList<Integer>>> matingPopulation;
75
76    /* @param allEdges
77     *           The complete set of all nodes of the graph that underlies the problem.
78     * @param allEdges
79     *           The complete set of all edges of the graph that underlies the problem.
80     * @param startNode
81     *           The node (normally the depot) at which the agent starts.
82     * @param tourNodes
83     *           The set of nodes the agent is supposed to visit.*/
84    public C report(List<V> allNodes, List<E> allEdges, V startNode,
85          List<V> tourNodes, V endNode, Function<V, Double> demandFunction,
86          Function<V, Double> earliestTimeFunction,
87          Function<V, Double> latestTimeFunction,
88          Function<V, Double> serviceTimeFunction, Function<E, V> startNodeFunction,
89          Function<E, V> endNodeFunction, Function<E, Double> edgeLengthFunction,
90          Function<E, Double> edgeTimeFunction,
91          Function<ArrayList<ArrayList<Pair<V, Boolean>>>, C> resultTourFunction,
92          double vehicleCapacity, int populationSize, double probabilityForGreedy,
93          double maxDistance, int generations, double weightNoOfTours,
94          double weightTotalDistance, int tournamentSize,
95          double takeBestFromTournamentProb, double mutationProb,
96          double latestAtDepot) throws ExtensionException, LogoException {
97
98       try {
99
100        startCity2 = startNode;
101
102        this.vehicleCapacity = vehicleCapacity;
103        this.endCity2 = endNode;
104
105        createAgentIntegerBidiMaps2(tourNodes, demandFunction, earliestTimeFunction,
106              latestTimeFunction, serviceTimeFunction);
107        createGraph2(allNodes, allEdges, startNodeFunction, endNodeFunction);
108
109        createCostMatrix2(edgeLengthFunction);
110
111        createTimeMatrix2(edgeTimeFunction);
112
113        generateInitialPopulation2(populationSize, probabilityForGreedy,
114              maxDistance);
115
116         // cf. Section 3.7 (Ombuki et al., 2006)
117         deriveRoutesFromChromosome2(latestAtDepot);
118
```

```
119          for (int i = 0; i < generations; i++) { /*Begin: main loop */
120             // cf. Section 2.2 (Ombuki et al., 2002)
121             rankChromosomes2();
122
123             // cf. Section 3.2.1 (Ombuki et al., 2006)
124             createWeightedSumSortedList2(weightNoOfTours, weightTotalDistance);
125
126             // cf. Section 3.4 (Ombuki et al., 2006)
127             generateMatingPopulation2(2 * (populationSize - 1), tournamentSize,
128                   takeBestFromTournamentProb);
129
130             // cf. Section 3.4 (elite model) and Section 3.5 (Recombination phase)
131             // in (Ombuki et al., 2006)
132             generateNewPopulation2(latestAtDepot);
133
134             // cf. Section 3.6 (Ombuki et al., 2006):
135             mutatePhenotypes2(mutationProb, latestAtDepot);
136
137             // utility
138             updateChromosomeDataStructures2(latestAtDepot);
139          }
140          // cf. Section 2.2 (Ombuki et al., 2002)
141          rankChromosomes2();
142
143          // cf. Section 3.2.1 (Ombuki et al., 2006)
144          createWeightedSumSortedList2(weightNoOfTours, weightTotalDistance);
145
146          return createReturnRoutes(edgeLengthFunction, startNodeFunction,
147                endNodeFunction, resultTourFunction);
148       } catch (Exception e) {
149          System.err.println("Something bad happened!");
150       }
151       return null;
152    }
153
154    private C createReturnRoutes(Function<E, Double> edgeLengthFunction,
155       Function<E, V> startNodeFunction, Function<E, V> endNodeFunction,
156       Function<ArrayList<ArrayList<Pair<V, Boolean>>>, C> resultTourFunction) {
157       Transformer<E, Double> edgeCostTransformer = l -> edgeLengthFunction.apply(l);
158
159       DijkstraShortestPath<V, E> dijkstra =
160         new DijkstraShortestPath<V, E>(graph2, edgeCostTransformer);
161
162       ArrayList<ArrayList<ArrayList<Integer>>> allTours =
163           new ArrayList<ArrayList<ArrayList<Integer>>>(population2);
164
165       allTours.sort((c1, c2) -> {
166          int rankComparison = Integer.compare(rankOfTours2.get(c1),
167                rankOfTours2.get(c2));
168          if (rankComparison != 0)
169             return rankComparison;
170          return Double.compare(weightedSumSortedMap.get(c1),
171                weightedSumSortedMap.get(c2));
172       });
173
174       ArrayList<ArrayList<Integer>> bestSolution = allTours.get(0);
175
176       ArrayList<ArrayList<V>> bestSolutionOriginal = new ArrayList<ArrayList<V>>();
177       ArrayList<V> bestSolutionTourOriginal = null;
178       for (ArrayList<Integer> integerTour : bestSolution) {
179          bestSolutionTourOriginal = new ArrayList<V>();
180          for (Integer i : integerTour) {
181             bestSolutionTourOriginal.add(agentIntegerBidiMap2.getKey(i));
182          }
183          bestSolutionOriginal.add(bestSolutionTourOriginal);
184       }
185
186       ArrayList<ArrayList<Pair<V, Boolean>>> bestSolutionWithStopIndication
187           = new ArrayList<ArrayList<Pair<V, Boolean>>>();
188
189       ArrayList<Pair<V, Boolean>> bTourWithStopIndication = null;
190
191       Pair<V, Boolean> customerWithStopIndication = null;
192
```

```
193         for (ArrayList<V> bTour : bestSolutionOriginal) {
194            bTourWithStopIndication = new ArrayList<Pair<V, Boolean>>();
195            V previousV = startCity2;
196            for (V v : bTour) {
197               List<E> routeOfEs = dijkstra.getPath(previousV, v);
198               V end1 = null;
199               for (E e : routeOfEs) {
200                  end1 = startNodeFunction.apply(e);
201                  if (end1.equals(previousV) && !end1.equals(startCity2))
202                     customerWithStopIndication = new Pair<V, Boolean>(end1, true);
203                  else
204                     customerWithStopIndication = new Pair<V, Boolean>(end1, false);
205                  bTourWithStopIndication.add(customerWithStopIndication);
206               }
207               previousV = v;
208            }
209            List<E> routeOfLinks = dijkstra.getPath(previousV, endCity2);
210            for (E e : routeOfLinks) {
211               V end1 = startNodeFunction.apply(e);
212               if (end1.equals(previousV))
213                  customerWithStopIndication = new Pair<V, Boolean>(end1, true);
214               else
215                  customerWithStopIndication = new Pair<V, Boolean>(end1, false);
216               bTourWithStopIndication.add(customerWithStopIndication);
217            }
218            customerWithStopIndication = new Pair<V, Boolean>(endCity2, true);
219            bTourWithStopIndication.add(customerWithStopIndication);
220            bestSolutionWithStopIndication.add(bTourWithStopIndication);
221         }
222         C res = resultTourFunction.apply(bestSolutionWithStopIndication);
223         return res;
224      }
225
226      private void createAgentIntegerBidiMaps2(List<V> tourNodes,
227            Function<V, Double> demandFunction,
228            Function<V, Double> earliestTimeFunction,
229            Function<V, Double> latestTimeFunction,
230            Function<V, Double> serviceTimeFunction) throws Exception {
231         agentIntegerBidiMap2 = new DualHashBidiMap<>();
232         if (!tourNodes.contains(startCity2)) {
233            agentIntegerBidiMap2.put(startCity2, 0);
234            for (int i = 0; i < tourNodes.size(); i++) {
235               agentIntegerBidiMap2.put(tourNodes.get(i), i + 1);
236            }
237         } else {
238            for (int i = 0; i < tourNodes.size(); i++) {
239               agentIntegerBidiMap2.put(tourNodes.get(i), i);
240            }
241         }
242
243         integerEarliestTimeMap2 = new HashMap<>();
244         integerLatestTimeMap2 = new HashMap<>();
245         integerServiceTimeMap2 = new HashMap<>();
246         integerDemandMap2 = new HashMap<>();
247         for (V node : agentIntegerBidiMap2.keySet()) {
248
249            Integer nodeIndex = agentIntegerBidiMap2.get(node);
250
251            this.integerEarliestTimeMap2.put(nodeIndex,
252                  earliestTimeFunction.apply(node));
253            this.integerLatestTimeMap2.put(nodeIndex, latestTimeFunction.apply(node));
254            this.integerServiceTimeMap2.put(nodeIndex, serviceTimeFunction.apply(node));
255            this.integerDemandMap2.put(nodeIndex, demandFunction.apply(node));
256         }
257      }
258
259      private void createGraph2(List<V> allNodes, List<E> allEdges,
260            Function<E, V> startNodeFunction, Function<E, V> endNodeFunction) {
261         graph2 = new DirectedSparseMultigraph<V, E>();
262         Iterator<V> nodeIterator = allNodes.iterator();
263         while (nodeIterator.hasNext())
264            graph2.addVertex(nodeIterator.next());
265         Iterator<E> edgeIterator = allEdges.iterator();
```

```
266         while (edgeIterator.hasNext()) {
267            E edge = edgeIterator.next();
268            graph2.addEdge(edge, startNodeFunction.apply(edge),
269                  endNodeFunction.apply(edge));
270         }
271      }
272
273      private void createCostMatrix2(Function<E, Double> linkLengthFunction) {
274         costMatrix2 = new double[agentIntegerBidiMap2.keySet()
275               .size()][agentIntegerBidiMap2.keySet().size()];
276         Transformer<E, Double> edgeCostTransformer = l -> linkLengthFunction.apply(l);
277         DijkstraShortestPath<V, E> dijkstra = new DijkstraShortestPath<V, E>(graph2,
278               edgeCostTransformer);
279
280         for (V srcNode : agentIntegerBidiMap2.keySet()) {
281            Map<V, Number> costMap = dijkstra.getDistanceMap(srcNode,
282                  new ArrayList<V>(agentIntegerBidiMap2.keySet()));
283            for (V tgtNode : costMap.keySet()) {
284               if (agentIntegerBidiMap2.containsKey(tgtNode)) {
285                  costMatrix2[agentIntegerBidiMap2.get(srcNode)][agentIntegerBidiMap2
286                        .get(tgtNode)] = costMap.get(tgtNode).doubleValue();
287               }
288            }
289         }
290      }
291
292      private void createTimeMatrix2(Function<E, Double> linkTimeFunction) {
293         timeMatrix2 = new double[agentIntegerBidiMap2.keySet()
294               .size()][agentIntegerBidiMap2.keySet().size()];
295         Transformer<E, Double> edgeTimeTransformer = l -> linkTimeFunction.apply(l);
296         DijkstraShortestPath<V, E> dijkstra = new DijkstraShortestPath<V, E>(graph2,
297               edgeTimeTransformer);
298         for (V srcNode : agentIntegerBidiMap2.keySet()) {
299            Map<V, Number> costMap = dijkstra.getDistanceMap(srcNode,
300                  new ArrayList<V>(agentIntegerBidiMap2.keySet()));
301            for (V tgtNode : costMap.keySet()) {
302               if (agentIntegerBidiMap2.containsKey(tgtNode)) {
303                  timeMatrix2[agentIntegerBidiMap2.get(srcNode)][agentIntegerBidiMap2
304                        .get(tgtNode)] = costMap.get(tgtNode).doubleValue();
305               }
306            }
307         }
308      }
309
310      /**
311       * IMPORTANT: A negative value represents an infeasible tour
312       *
313       * @return The length of the tour. A negative value indicates an infeasible
314       *         tour.
315       */
316      private double getLengthOfTour2(ArrayList<Integer> tour, double latestAtDepot) {
317         double length = costMatrix2[agentIntegerBidiMap2.get(startCity2)][tour.get(0)];
318         double arrival = timeMatrix2[agentIntegerBidiMap2.get(startCity2)][tour.get(0)];
319         double leave = 0.0;
320         double currentCapacityUsage = integerDemandMap2.get(tour.get(0));
321         double customerEarliestArrival = integerEarliestTimeMap2.get(tour.get(0));
322         double customerLatestArrival = integerLatestTimeMap2.get(tour.get(0));
323         double customerServiceTime = integerServiceTimeMap2.get(tour.get(0));
324         double timeBackToEndDepot = timeMatrix2[tour.get(0)][agentIntegerBidiMap2.get(endCity2)];
325
326         double timeBackAtEndDepot = Math.max(arrival, customerEarliestArrival)
327               + customerServiceTime + timeBackToEndDepot;
328
329         if ((customerLatestArrival > 0.001 && arrival > customerLatestArrival)
330               || currentCapacityUsage > vehicleCapacity
331               || timeBackAtEndDepot > latestAtDepot)
332            return -1.0; // tour is infeasible
333
334         if (arrival < customerEarliestArrival) // then the vehicle has to wait
335            arrival = customerEarliestArrival;
336
337         leave = arrival + customerServiceTime;
338
```

```
339         for (int i = 1; i < tour.size(); i++) {
340            length = length + costMatrix2[tour.get(i - 1)][tour.get(i)];
341            arrival = leave + timeMatrix2[tour.get(i - 1)][tour.get(i)];
342            customerEarliestArrival = integerEarliestTimeMap2.get(tour.get(i));
343            customerLatestArrival = integerLatestTimeMap2.get(tour.get(i));
344            currentCapacityUsage = currentCapacityUsage
345                  + integerDemandMap2.get(tour.get(i));
346            timeBackToEndDepot = timeMatrix2[tour.get(i)][agentIntegerBidiMap2
347                  .get(endCity2)];
348            timeBackAtEndDepot = Math.max(arrival, customerEarliestArrival)
349                  + customerServiceTime + timeBackToEndDepot;
350
351            if ((customerLatestArrival > 0.001 && arrival > customerLatestArrival)
352                  || currentCapacityUsage > vehicleCapacity
353                  || timeBackAtEndDepot > latestAtDepot) {
354               return -1.0d; // tour is infeasible
355            }
356            if (arrival < customerEarliestArrival) {
357               arrival = customerEarliestArrival;
358            }
359            leave = arrival + customerServiceTime;
360         }
361
362         length = length + costMatrix2[tour.get(tour.size() - 1)][agentIntegerBidiMap2
363               .get(endCity2)];
364         return length;
365      }
366
367      /**
368       * Utility method.
369       *
370       * @return The total distance covered by all vehicles. IMPORTANT: A negative
371       *         value represents an infeasible tour.
372       */
373      private double getTotalLengthForTours2(ArrayList<ArrayList<Integer>> tours,
374            double latestAtDepot) {
375         double totalLength = 0.0;
376         double tourLength = 0.0;
377         for (ArrayList<Integer> tour : tours) {
378            tourLength = getLengthOfTour2(tour, latestAtDepot);
379            if (tourLength < 0)
380               return -1;
381            else
382               totalLength = totalLength + tourLength;
383         }
384         return totalLength;
385      }
386
387      private void updateChromosomeDataStructures2(double latestAtDepot) {
388         for (ArrayList<ArrayList<Integer>> tours : population2)
389            toursTotalLength2.put(tours, getTotalLengthForTours2(tours, latestAtDepot));
390      }
391
392      /** cf. Section 2.2 (Ombuki et al., 2006) */
393      private void rankChromosomes2() {
394         rankOfTours2 = new HashMap<ArrayList<ArrayList<Integer>>, Integer>();
395         ArrayList<ArrayList<ArrayList<Integer>>> populationCopy
396             = new ArrayList<ArrayList<ArrayList<Integer>>>(population2);
397         int currentRank = 1;
398         double currentToursLength;
399         double currentToursVehicles;
400         double otherToursLength;
401         double otherToursVehicles;
402         ArrayList<ArrayList<ArrayList<Integer>>> toursOfCurrentRank = new ArrayList<>();
403
404         while (!populationCopy.isEmpty()) {
405            for (ArrayList<ArrayList<Integer>> currentTours : populationCopy) {
406               boolean dominated = false;
407               currentToursLength = toursTotalLength2.get(currentTours);
408               currentToursVehicles = currentTours.size();
409               Iterator<ArrayList<ArrayList<Integer>>> checkSet = populationCopy.iterator();
410               ArrayList<ArrayList<Integer>> otherTours;
```

```
411             while (!dominated && checkSet.hasNext()) {
412                otherTours = checkSet.next();
413                otherToursLength = toursTotalLength2.get(otherTours);
414                otherToursVehicles = otherTours.size();
415
416                dominated = otherToursLength < currentToursLength
417                      && otherToursVehicles <= currentToursVehicles
418                      || otherToursLength <= currentToursLength
419                            && otherToursVehicles < otherToursVehicles;
420             }
421             if (!dominated) {
422                rankOfTours2.put(currentTours, currentRank);
423                toursOfCurrentRank.add(currentTours);
424             }
425          }
426          populationCopy.removeAll(toursOfCurrentRank);
427          toursOfCurrentRank.clear();
428          currentRank++;
429       }
430    }
431
432    /** cf. Section 3.1 (Ombuki et al., 2006) */
433    private void generateInitialPopulation2(int populationSize,
434          double probabilityForGreedy, double maxDistance) throws ExtensionException {
435
436       ArrayList<Integer> originalTourNodes = new ArrayList<>(
437             agentIntegerBidiMap2.values());
438       ArrayList<Integer> copyTourNodes;
439
440       if (originalTourNodes.contains(agentIntegerBidiMap2.get(startCity2)))
441          originalTourNodes.remove(agentIntegerBidiMap2.get(startCity2));
442
443       chromosomes2 = new Integer[populationSize][originalTourNodes.size()];
444
445       for (int i = 0; i < populationSize; i++) {
446          if (Math.random() > probabilityForGreedy) {
447             copyTourNodes = new ArrayList<>(originalTourNodes);
448             for (int j = 0; j < originalTourNodes.size(); j++)
449                chromosomes2[i][j] = copyTourNodes
450                      .remove((int) (Math.random() * copyTourNodes.size()));
451          } else {
452             copyTourNodes = new ArrayList<>(originalTourNodes);
453             int ci = 0;
454             chromosomes2[i][ci] = copyTourNodes
455                   .remove((int) (Math.random() * copyTourNodes.size()));
456             while (ci < originalTourNodes.size() - 1) {
457                double currentDistance = Double.MAX_VALUE;
458                for (Integer node : copyTourNodes) {
459                   double distanceTo_node = costMatrix2[chromosomes2[i][ci]][node];
460                   if (distanceTo_node < maxDistance
461                         && distanceTo_node < currentDistance) {
462                      currentDistance = distanceTo_node;
463                      chromosomes2[i][ci + 1] = node;
464                   }
465                }
466                if (chromosomes2[i][ci + 1] != null) { // found a node within the node barrier
467                   copyTourNodes.remove(chromosomes2[i][ci + 1]);
468                } else {
469                   chromosomes2[i][ci + 1] = copyTourNodes
470                         .remove((int) (Math.random() * copyTourNodes.size()));
471                }
472                ci = ci + 1;
473             }
474          }
475       }
476    }
477
```

```
478    /** // cf. Section 3.7 (Ombuki et al., 2006) */
479    private void deriveRoutesFromChromosome2(double latestAtDepot) {
480       population2 = new ArrayList<ArrayList<ArrayList<Integer>>>();
481       toursTotalLength2 = new HashMap<ArrayList<ArrayList<Integer>>, Double>();
482
483       try {
484          for (int i = 0; i < chromosomes2.length; i++) {
485             double length = 0.0;
486             double arrivalTime = 0.0;
487             double leaveTime = 0.0;
488             double currentCapacityUsage = 0.0;
489             int numberOfCustomersOfCurrentTour = 0;
490
491             ArrayList<ArrayList<Integer>> listOfTours = new ArrayList<>();
492             population2.add(listOfTours);
493             ArrayList<Integer> currentTour = null;
494
495             for (int j = 0; j < chromosomes2[i].length; j++) {
496                if (numberOfCustomersOfCurrentTour == 0) { // we're dealing with the
497                                                           // first customer of a tour
498                   currentTour = new ArrayList<>();
499                   listOfTours.add(currentTour);
500                   length = length + costMatrix2[agentIntegerBidiMap2
501                         .get(startCity2)][chromosomes2[i][j]];
502
503                   arrivalTime = timeMatrix2[agentIntegerBidiMap2
504                         .get(startCity2)][chromosomes2[i][j]];
505
506                   double customerDemand = integerDemandMap2.get(chromosomes2[i][j]);
507                   double customerEarliestTime = integerEarliestTimeMap2
508                         .get(chromosomes2[i][j]);
509                   double customerLatestTime = integerLatestTimeMap2
510                         .get(chromosomes2[i][j]);
511                   double customerServiceTime = integerServiceTimeMap2
512                         .get(chromosomes2[i][j]);
513
514                   double timeBackToEndDepot
515                     = timeMatrix2[chromosomes2[i][j]][agentIntegerBidiMap2.get(endCity2)];
516
517                   double timeBackAtEndDepot = Math.max(arrivalTime,
518                         customerEarliestTime) + customerServiceTime
519                         + timeBackToEndDepot;
520
521                   if ((customerLatestTime > 0.001
522                         && arrivalTime > customerLatestTime)
523                         || customerDemand > vehicleCapacity) {
524                           throw new RuntimeException("Problem infeasible");
525                   }  else if (timeBackAtEndDepot > latestAtDepot) {
526                           throw new RuntimeException("Problem infeasible");
527                   } else {
528                      LOG.trace("Problem is not infeasible.");
529                      currentTour.add(chromosomes2[i][j]);
530                      numberOfCustomersOfCurrentTour++;
531                      currentCapacityUsage = customerDemand;
532                      if (arrivalTime < customerEarliestTime)
533                         arrivalTime = customerEarliestTime;
534                      leaveTime = arrivalTime + customerServiceTime;
535                   }
536                } else { // we're not dealing with the first customer of a tour (i.e.
537                         // we're in the middle of a tour)
538
539                   double customerDemand = integerDemandMap2.get(chromosomes2[i][j]);
540
541                   currentCapacityUsage = currentCapacityUsage + customerDemand;
542                   arrivalTime = leaveTime
543                         + timeMatrix2[chromosomes2[i][j - 1]][chromosomes2[i][j]];
544
545                   double customerEarliestArrival = integerEarliestTimeMap2.get(chromosomes2[i][j]);
546
547                   double customerLatestArrival = integerLatestTimeMap2.get(chromosomes2[i][j]);
548
549                   double serviceTime = integerServiceTimeMap2.get(chromosomes2[i][j]);
550
551                   double timeBackToEndDepot =
552                       timeMatrix2[chromosomes2[i][j]][agentIntegerBidiMap2.get(endCity2)];
553
```

```
554                 double timeBackAtEndDepot = Math.max(arrivalTime, customerEarliestArrival)
555                     + serviceTime + timeBackToEndDepot;
556
557                 if ((customerLatestArrival > 0.001
558                       && arrivalTime > customerLatestArrival)
559                       || currentCapacityUsage > vehicleCapacity
560                       || timeBackAtEndDepot > latestAtDepot) {
561                    length = length
562                        + costMatrix2[chromosomes2[i][j - 1]][agentIntegerBidiMap2.get(endCity2)];
563                    j = j - 1;
564                    numberOfCustomersOfCurrentTour = 0;
565                 } else {
566                    length = length
567                          + costMatrix2[chromosomes2[i][j - 1]][chromosomes2[i][j]];
568                    currentTour.add(chromosomes2[i][j]);
569                    numberOfCustomersOfCurrentTour++;
570                    if (arrivalTime < customerEarliestArrival)
571                       arrivalTime = customerEarliestArrival;
572                    leaveTime = arrivalTime + serviceTime;
573                 }
574              }
575              if (j == chromosomes2[i].length - 1) {
576                 length = length
577                       + costMatrix2[chromosomes2[i][j]][agentIntegerBidiMap2
578                             .get(endCity2)];
579                 toursTotalLength2.put(listOfTours, length);
580              }
581           }
582        } // end of phase 1 of derivement lists of routes from chromosomes
583     } catch (Exception e) { }
584
585     for (ArrayList<ArrayList<Integer>> tours : population2) {
586        Iterator<ArrayList<Integer>> iterator = tours.iterator();
587        ArrayList<Integer> firstTour = iterator.next();
588        while (iterator.hasNext()) {
589           ArrayList<Integer> secondTour = iterator.next();
590           Double lengthOfFirst = getLengthOfTour2(firstTour, latestAtDepot);
591           Double lengthOfSecond = getLengthOfTour2(secondTour, latestAtDepot);
592           if (lengthOfFirst < 0 || lengthOfSecond < 0)
593              throw new RuntimeException(
594                    "Should not happen, since tours were ensured to be feasible");
595           Double sumFirst = lengthOfFirst + lengthOfSecond;
596
597           // switch customer according to the rules stated by (Ombuki et al., 2006)
598           if (firstTour.size() > 1)
599              secondTour.add(0, firstTour.remove(firstTour.size() - 1));
600           else
601              continue;
602           lengthOfFirst = getLengthOfTour2(firstTour, latestAtDepot);
603           lengthOfSecond = getLengthOfTour2(secondTour, latestAtDepot);
604           Double sumSecond = lengthOfFirst + lengthOfSecond;
605
606           if (lengthOfFirst < 0 || lengthOfSecond < 0 || (sumSecond > sumFirst))
607              firstTour.add(secondTour.remove(0));
608           else
609              toursTotalLength2.put(tours,
610                    getTotalLengthForTours2(tours, latestAtDepot));
611           firstTour = secondTour;
612        }
613     }
614  }
615  /** cf. Section 3.2.1 (Ombuki et al. 2006) */
616  private void createWeightedSumSortedList2(double weightNumberOfTours,
617        double weightTotalDistance) {
618     weightedSumSortedMap = new HashMap<ArrayList<ArrayList<Integer>>, Double>();
619     weightOrderedSumSortedList2 = new ArrayList<>(population2);
620     weightOrderedSumSortedList2.forEach(it -> {
621        weightedSumSortedMap.put(it, weightNumberOfTours * it.size()
622              + weightTotalDistance * toursTotalLength2.get(it));
623     });
624     weightOrderedSumSortedList2.sort((c1, c2) -> {
625        return Double.compare(weightedSumSortedMap.get(c1),
626              weightedSumSortedMap.get(c2));
627     });
628  }
629
```

```
630     /** cf. Section 3.4 (Ombuki et al., 2006) */
631     private void generateMatingPopulation2(int matingPopulationSize,
632           int tournamentSize, double takeBestOfTournamentProb) {
633
634        this.matingPopulation = new ArrayList<ArrayList<ArrayList<Integer>>>();
635        ArrayList<ArrayList<ArrayList<Integer>>> tournamentSet = new ArrayList<>();
636
637        while (matingPopulation.size() < matingPopulationSize) {
638           while (tournamentSet.size() < tournamentSize) {
639              ArrayList<ArrayList<Integer>> participant = population2
640                    .get((int) (Math.random() * population2.size()));
641              tournamentSet.add(participant);
642           }
643           if (Math.random() < takeBestOfTournamentProb) {
644              tournamentSet.sort((c1, c2) -> {
645                 int rankResult = Integer.compare(rankOfTours2.get(c1),
646                       rankOfTours2.get(c2));
647                 if (rankResult != 0)
648                    return rankResult;
649                 return Double.compare(weightedSumSortedMap.get(c1),
650                       weightedSumSortedMap.get(c2));
651              });
652              matingPopulation.add(tournamentSet.get(0));
653           } else {
654              tournamentSet.forEach(it -> matingPopulation.add(it));
655           }
656           tournamentSet.clear();
657        }
658     }
659
660     /* cf. Section 3.4 (elite model) and Section 3.5 (Recombination phase) in
661      * (Ombuki et al., 2006) */
662     private void generateNewPopulation2(double latestAtDepot) {
663        ArrayList<ArrayList<ArrayList<Integer>>> populationNextGen
664            = new ArrayList<ArrayList<ArrayList<Integer>>>();
665        ArrayList<ArrayList<ArrayList<Integer>>> allTours = new ArrayList<>(
666              population2);
667        allTours.sort((c1, c2) -> {
668           int rankResult = Integer.compare(rankOfTours2.get(c1),
669                 rankOfTours2.get(c2));
670           if (rankResult != 0)
671              return rankResult;
672           return Double.compare(weightedSumSortedMap.get(c1),
673                 weightedSumSortedMap.get(c2));
674        });
675        ArrayList<ArrayList<Integer>> theChampion = allTours.get(0);
676        populationNextGen.add(theChampion);
677
678        while (matingPopulation.size() > 2) {
679           ArrayList<ArrayList<Integer>> parent1 = matingPopulation.remove(0);
680           ArrayList<ArrayList<Integer>> parent2 = matingPopulation.remove(0);
681
682           ArrayList<ArrayList<Integer>> toursForParent1 = new ArrayList<>();
683           for (ArrayList<Integer> tourList : parent1)
684              toursForParent1.add(new ArrayList<>(tourList));
685
686           ArrayList<ArrayList<Integer>> toursForParent2 = new ArrayList<>();
687           for (ArrayList<Integer> tourList : parent2)
688              toursForParent2.add(new ArrayList<>(tourList));
689
690           ArrayList<Integer> removeTourOfParent1 = new ArrayList<>(
691                 toursForParent1.get((int) (Math.random() * toursForParent1.size())));
692           ArrayList<Integer> removeTourOfParent2 = new ArrayList<>(
693                 toursForParent2.get((int) (Math.random() * toursForParent2.size())));
694
695           for (int i = 0; i < removeTourOfParent1.size(); i++)
696              for (int j = 0; j < toursForParent2.size(); j++)
697                 if (toursForParent2.get(j).contains(removeTourOfParent1.get(i)))
698                    toursForParent2.get(j).remove(removeTourOfParent1.get(i));
699           // remove empty tours
700           Iterator<ArrayList<Integer>> toursOfParent2Iterator = toursForParent2
701                 .iterator();
702           while (toursOfParent2Iterator.hasNext())
703              if (toursOfParent2Iterator.next().isEmpty())
704                 toursOfParent2Iterator.remove();
705
```

```
706          for (int i = 0; i < removeTourOfParent2.size(); i++)
707             for (int j = 0; j < toursForParent1.size(); j++)
708                if (toursForParent1.get(j).contains(removeTourOfParent2.get(i)))
709                   toursForParent1.get(j).remove(removeTourOfParent2.get(i));
710          // remove empty tours
711          Iterator<ArrayList<Integer>> toursOfParent1Iterator = toursForParent1.iterator();
712          while (toursOfParent1Iterator.hasNext())
713             if (toursOfParent1Iterator.next().isEmpty())
714                toursOfParent1Iterator.remove();
715
716          // Now the removed customers are to be replaced at the optimal places
717          while (!removeTourOfParent1.isEmpty()) {
718             Integer customer2BeInserted = removeTourOfParent1
719                   .remove((int) (Math.random() * removeTourOfParent1.size()));
720             int best_i = -1; // best tour to place the customer
721             int best_j = -1; // best spot inside the tour
722             double shortestKnownPossibleTour = Double.MAX_VALUE;
723             for (int i = 0; i < toursForParent2.size(); i++) {
724                ArrayList<Integer> currentTour = toursForParent2.get(i);
725                int currentTourSize = currentTour.size();
726                for (int j = 0; j < currentTourSize; j++) {
727                   currentTour.add(j, customer2BeInserted);
728                   double totallengthOfCurrentTours = getTotalLengthForTours2(
729                         toursForParent2, latestAtDepot);
730                   if (totallengthOfCurrentTours > 0 // negative length indicates infeasible tour
731                         && totallengthOfCurrentTours < shortestKnownPossibleTour) {
732                      best_i = i;
733                      best_j = j;
734                      shortestKnownPossibleTour = totallengthOfCurrentTours;
735                   }
736                   currentTour.remove(j);// perhaps we'll find a(n even) better place
737                }
738             }
739             if (best_i == -1) // customer could not be inserted anywhere
740                toursForParent2.add( new ArrayList<Integer>(Arrays.asList(customer2BeInserted)));
741             else
742                toursForParent2.get(best_i).add(best_j, customer2BeInserted);
743
744             populationNextGen.add(toursForParent2);
745          }
746          // Now the removed customers are to be replaced at the optimal places
747          while (!removeTourOfParent2.isEmpty()) {
748             Integer customer2BeInserted = removeTourOfParent2
749                   .remove((int) (Math.random() * removeTourOfParent2.size()));
750             int best_i = -1; // best tour to place the customer
751             int best_j = -1; // best spot inside the tour
752             double shortestKnownPossibleTour = Double.MAX_VALUE;
753             for (int i = 0; i < toursForParent1.size(); i++) {
754                ArrayList<Integer> currentTour = toursForParent1.get(i);
755                int currentTourSize = currentTour.size();
756                for (int j = 0; j < currentTourSize; j++) {
757                   currentTour.add(j, customer2BeInserted);
758                   double lengthOfCurrentTour = getTotalLengthForTours2(
759                         toursForParent1, latestAtDepot);
760                   // negative length indicates infeasible tour
761                   if (lengthOfCurrentTour > 0
762                         && lengthOfCurrentTour < shortestKnownPossibleTour) {
763                      best_i = i;
764                      best_j = j;
765                      shortestKnownPossibleTour = lengthOfCurrentTour;
766                   }
767                   // perhaps we'll find a(n even) better place :-D
768                   currentTour.remove(j);
769                }
770             }
771             if (best_i == -1) // could customer could not be inserted anywhere
772                toursForParent1.add(
773                      new ArrayList<Integer>(Arrays.asList(customer2BeInserted)));
774             else
775                toursForParent1.get(best_i).add(best_j, customer2BeInserted);
776
777             populationNextGen.add(toursForParent1);
778          }
779       }
780       population2 = populationNextGen;
781    }
```

```
782     /** // cf. Section 3.6 (Ombuki et al., 2006): */
783     private void mutatePhenotypes2(double chanceForMutation, double latestAtDepot) {
784        for (ArrayList<ArrayList<Integer>> tours : population2) {
785           if (Math.random() < chanceForMutation) {
786              ArrayList<Integer> tourToBeMutated = tours
787                    .get((int) (Math.random() * tours.size()));
788              int mutationIndex = (int) (Math.random() * tourToBeMutated.size());
789              if ((mutationIndex + 1) < tourToBeMutated.size()) {
790                 Integer customerToBeSwapped = tourToBeMutated.remove(mutationIndex);
791                 tourToBeMutated.add(mutationIndex + 1, customerToBeSwapped);
792                 if (getLengthOfTour2(tourToBeMutated, latestAtDepot) < 0)
793                    tourToBeMutated.add(mutationIndex,
794                          tourToBeMutated.remove(mutationIndex + 1));
795              }
796           }
797        }
798     }
799  }
```

B.2 Invocation

Listing B.2: Invocation of Ombukis evolutionary algorithm.

```
// V -> node type (vertices)
// E -> edge type
// C -> Collection type for return collection
public class NetLogoVRPTWOmbuki implements Reporter {

   public Syntax getSyntax() {
      return SyntaxJ.reporterSyntax(new int[] {
            Syntax.TurtlesetType(),   // args[0] // cities
            Syntax.LinksetType(),     // args[1] edges
            Syntax.TurtleType(),      // args[2] startCity
            Syntax.ListType(),        // args[3] cities of the tour
            Syntax.NumberType(),      // args[4] HERE: vehicle capacity!
            Syntax.NumberType(),      // args[5] populationSize
            Syntax.NumberType(),      // args[6] probabilityForGreedy
            Syntax.NumberType(),      // args[7] maxDistance
            Syntax.NumberType(),      // args[8] generations
            Syntax.NumberType(),      // args[9] weigthNoOfTours
            Syntax.NumberType(),      // args[10] weightTotalDistance
            Syntax.NumberType(),      // args[11] tournamentSize
            Syntax.NumberType(),      // args[12] takeBestFromTournament
            Syntax.NumberType(),      // args[13] mutationProb
            Syntax.TurtleType()       // args[14] endCity
      }, Syntax.ListType());
   }

   public Object report(Argument args[], Context context)
         throws ExtensionException, LogoException {

      org.nlogo.agent.Turtle startTurtle = (org.nlogo.agent.Turtle) args[2].getTurtle();

      org.nlogo.agent.World aWorld = (org.nlogo.agent.World) startTurtle.world();
      HashMap<String, Integer> breedsOwnCache = aWorld.breedsOwnCache();

      NetLogoToAthosVRPTWNode adaptedStartTurtle = new NetLogoToAthosVRPTWNode(startTurtle);

      VRPTWSolverOmbuki<Turtle, Link, LogoList> ombukiSolver = new VRPTWSolverOmbuki<>();

      // 0
      org.nlogo.agent.AgentSet cities = (org.nlogo.agent.AgentSet) args[0].getAgentSet();
      List<Turtle> allCities = new ArrayList<>();
      AgentIterator turtleIterator = cities.iterator();
      while (turtleIterator.hasNext()) {
         allCities.add((Turtle) turtleIterator.next());
      }

      // 1
      org.nlogo.agent.AgentSet links = (org.nlogo.agent.AgentSet) args[1].getAgentSet();
      List<Link> allEdges = new ArrayList<>();
      AgentIterator linkIterator = links.iterator();
      while (linkIterator.hasNext()) {
         allEdges.add((Link) linkIterator.next());
      }

      // 2
      org.nlogo.agent.Turtle startCity = (org.nlogo.agent.Turtle) args[2]
            .getTurtle();

      // 3
      LogoList customersToServe = args[3].getList();
      List<Turtle> tourCustomers = new ArrayList<>();
      scala.collection.Iterator<Object> tourTurtleIterator = customersToServe
            .iterator();
      while (tourTurtleIterator.hasNext()) {
         tourCustomers.add((Turtle) tourTurtleIterator.next());
      }

      // 4
      Turtle endCity = args[14].getTurtle();
      // 5
```

```
70        Function<Turtle, Double> demandFunction = (Turtle t) -> Double
71              .parseDouble(t.getVariable(25).toString());
72
73        // 6
74        Function<Turtle, Double> earliestTimeFunction = (Turtle t) -> Double
75              .parseDouble(t.getVariable(22).toString());
76
77        // 7
78        Function<Turtle, Double> latestTimeFunction = (Turtle t) -> Double
79              .parseDouble(t.getVariable(23).toString());
80
81        // 8
82        Function<Turtle, Double> serviceTimeFunction = (Turtle t) -> Double
83              .parseDouble(t.getVariable(24).toString());
84
85        // 9
86        Function<Link, Turtle> startNodeFunction = (Link l) -> l.end1();
87
88        // 10
89        Function<Link, Turtle> endNodeFunction = (Link l) -> l.end2();
90
91        // 11
92        Function<Link, Double> edgeLengthFunction = (Link l) -> Double
93              .parseDouble(l.getVariable(13).toString());
94
95        // 12
96        Function<Link, Double> edgeTimeFunction = (Link l) -> Double
97              .parseDouble(l.getVariable(16).toString());
98
99        // 13
100       Function<ArrayList<ArrayList<Pair<Turtle, Boolean>>>, LogoList> resultListFunction = (
101             ArrayList<ArrayList<Pair<Turtle, Boolean>>> sourceList) -> {
102          LogoListBuilder outerMap = new LogoListBuilder();
103          LogoListBuilder innerMap = null;
104          LogoListBuilder tuple = null;
105          for (ArrayList<Pair<Turtle, Boolean>> tour : sourceList) {
106             innerMap = new LogoListBuilder();
107             for (Pair<Turtle, Boolean> pair : tour) {
108                tuple = new LogoListBuilder();
109                tuple.add(pair.getFirst());
110                tuple.add(pair.getSecond());
111                innerMap.add(tuple.toLogoList());
112             }
113             outerMap.add(innerMap.toLogoList());
114          }
115          return outerMap.toLogoList();
116       };
117
118       double vehicleCapacity = args[4].getDoubleValue();
119       int populationSize = args[5].getIntValue();
120       double probabilityForGreedy = args[6].getDoubleValue();
121       double maxDistance = args[7].getDoubleValue();
122       int generations = args[8].getIntValue();
123       double weightNoOfTours = args[9].getDoubleValue();
124       double weightTotalDistance = args[10].getDoubleValue();
125       int tournamentSize = args[11].getIntValue();
126       double takeBestFromTournamentProb = args[12].getDoubleValue();
127
128       double mutationProb = args[13].getDoubleValue();
129       double latestAtDepot = Double.parseDouble(endCity.getVariable(23).toString());
130       latestAtDepot = latestAtDepot == 0 ? Double.MAX_VALUE : latestAtDepot;
131
132       return ombukiSolver.report(allCities, allEdges, startCity, tourCustomers,
133             endCity, demandFunction, earliestTimeFunction, latestTimeFunction,
134             serviceTimeFunction, startNodeFunction, endNodeFunction,
135             edgeLengthFunction, edgeTimeFunction, resultListFunction,
136             vehicleCapacity, populationSize, probabilityForGreedy, maxDistance,
137             generations, weightNoOfTours, weightTotalDistance, tournamentSize,
138             takeBestFromTournamentProb, mutationProb, latestAtDepot);
139
140    }
141 }
```

Appendix C

Survey tasks

This section provides material from the controlled experiments discussed in Chapter 6 and Chapter 7. Section C.1 shows the informed consent form that all participants agreed to. Section C.2 and Section C.3 show the tasks participants had to solve with Athos and JSprit, respectively. For each task the correct solution is given together with the number of points awarded for correct answers as well as the number of points deducted for wong and missing answers. The parameters of the point scheme applied to each question are explained in Table C.1.

Table C.1: Parameters of the point scheme.

Parameter	Description
Answer Cell	Indicates whether this question was evaluated automatically with a VBA script that evaluates a given answer stored in a table cell.
CorAnswersString	Indicates whether this question was evaluated automatically with a VBA script that compares a given answer to an expected string of correct answers.
Points	Number of points awarded for a correct answer (for many questions multiple correct answers exist).
WrgAnswersString	Indicates whether there is a set of wrong answers that when given by a participant lead to the deduction of points.
NegPointsP	Number of points deducted for a) a missing correct answer, b) a wrong answer found in the WrgAnswersString
WrongAnswAllowed	Number of wrong answers that are allowed without deduction of points.
MissingAnswAllowed	Number of missing answers that are allowed without deduction of points.
MaxPointsToGet	The maximum number of points awarded for the respective task.

The material was provided in the form of HTML encoded pages created with the NoviSurvey software[1]. The design from the survey was adapted to the format of this thesis. The texts and tasks were adopted from the survey without changes.

[1] `https://novisurvey.net/`

C.1 Informed consent form

Task: Informed consent form

Informed consent

Edinburgh Napier University requires that all persons who participate in research studies give their written consent to do so. Please read the following and sign it if you agree with what it says.

1. I freely and voluntarily consent to be a participant in this study that evaluates the domain-specific language (DSL) Athos. Athos is a language designed for the domain of traffic and transport simulation and optimisation. The study is conducted by Benjamin Hoffmann, a postgraduate student at Edinburgh Napier University.

2. The broad goal of this research study is to gain insight into several language aspects (e.g. learnability, perceivability, evolvability). This study is conducted among students from THM enrolled for the *<NAME OF THE STUDY COURSE>*. I will be asked to provide a brief overview on my educational background and my prior programming experience before solving a set of tasks from the domain of vehicle routing problems. I understand that these information are only used for scientific purpose (e.g. to gain insight on how prior knowledge affects the learnability of Athos). In order to solve these tasks I am supposed to use both the aforementioned DSL and a GPL together with an application library called "JSprit". Completion of the survey should take no longer than 60 minutes.

3. I understand that results from this study may likely be published in various forms of scientific literature (journal articles, conference papers, PhD thesis, etc.).

4. I have been told that my responses will be anonymised. My name will not be linked with the research materials, and I will not be identified or identifiable in any report subsequently produced by the researcher.

5. I understand that if at any time during the survey I feel unable or unwilling to continue, I am free to leave. That is, my participation in this study is completely voluntary, and I may withdraw from it without negative consequences. However, I understand that I submit my answers anonymously. Therefore, once I press the "submit answers button" it is not possible to remove any of my answers from the set of collected answers.

6. Should I not wish to answer any particular question or questions, I am free to decline. For this purpose, each question features an op-out answer (e.g. "not applicable") which I may select without provision of further explanation.

7. I have been given the opportunity to ask questions regarding the survey and my questions have been answered to my satisfaction.

8. I have read and understand the above and consent to participate in this study. My agreement (given by checking the agreement checkbox and clicking the "start button") is not a waiver of any legal rights. Furthermore, I understand that I will be handed a copy of the informed consent form for my records upon request.

Do you agree with these terms and conditions?

☐ I agree with these terms and conditions.

C.2 Athos tasks

Task: Q01ATNW

Introduction

In this task you are given a program that contains a few syntactical mistakes.Your task is to spot and report these mistakes.

Task

☐ Line 1	☐ Line 2	☐ Line 3	☐ Line 4
☐ Line 5	☐ Line 6	☐ Line 7	☐ Line 8
☐ Line 9	☐ Line 10	☐ Line 11	☐ Line 12
☐ Line 13	☐ Line 14	☐ Line 15	☐ Line 16
☐ Line 17	☐ Line 18		

```
1  model q1nw
2  products
3     stuff ("quantity") 1.0
4  functions
5     calculate roadFunction  4 * length + 4
6     durationFunction highwayFunction  2 * length + 2
7  network
8     nodes
9        n0 at (5, 1)
10       n1 locatedAt (10, 9)
11       n2 at (12, 1)
12    edges
13       group beltway function highwayFunction members
14          b1 from n0 to n1
15          from n0 to n2
16       group road  function roadFunction members
17          r1 starts at node n1 ends at node n2
18          r2 from n2 to n1
```

Correct solution

☑Line 3, ☑Line 5, ☑Line 10, ☑Line 15, ☑Line 17

Evaluation 2020

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	2
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Evaluation 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q01ATAG

Introduction

In this task you are given a program that contains a few syntactical mistakes.Your task is to spot and report these mistakes.

Task

☐ Line 1	☐ Line 2	☐ Line 3	☐ Line 4
☐ Line 5	☐ Line 6	☐ Line 7	☐ Line 8
☐ Line 9	☐ Line 10		

```
1  agentTypes
2    agentType delivery1 maxWeight 360
3      behaviour awt awaitTourExternal when finished do die
4      behaviour die disappear
5    agentType delivery   2 maxWeight 360
6      behaviour awt awaitTourExternal when finished do die
7      behaviour die vanish
8    agent delivery3 maxWeight 360
9      behaviour awt awaitTourExternal when finished do "die"
10     behaviour die vanish
```

Correct solution

☑Line 4, ☑Line 5,☑Line 8, ☑Line 9

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q02ATAG (1/2)

Introduction

In this task you will find a program with a gap. Additionally, you are presented four code snippets that can be used to fill the gap. However, some of these snippets do not make sense (either for themselves or in the completed program). It is your task to find the nonsensical snippets and report them.

Task

Below you see a VRPTW modelled with Athos. In line 35, the definition of a depot is required. Which of the proposed snippets given at the bottom of this page are semantically incorrect (in other words: which of the four snippets do not make complete sense)?

☐ Snippet 1	☐ Snippet 2	☐ Snippet 3	☐ Snippet 4

```
1  model newModel
2    products
3      stuff weight 30.0
4    functions
5      durationFunction roadFunction length + cfactor
6    network
7      nodes
8        n0 (1.0, 1.0)
9        n1 (1.0, 8.0)   hasDemand stuff units 30.0
10       n2 (2.0, 11.0)  hasDemand stuff units 30.0
11       n3 (4.0, 6.0)
12       n4 (5.0, 12.0)  hasDemand stuff units 30.0
13       n5 (8.0, 11.0)  hasDemand stuff units 30.0
14       n6 (8.0, 7.0)   hasDemand stuff units 30.0
15       n7 (13.0, 12.0)
16       n8 (9.0, 5.0)
17       n9 (13.0, 1.0)
18     edges
19       group lcfgroup cfactor 2.0 function roadFunction  members
20         el01 from n0 to n1
21         el02 from n1 to n2
22         el03 from n2 to n4
23         el04 from n4 to n5
24         el05 from n6 to n5
25         el06 from n7 to n4
26         el07 from n7 to n9
27         el08 from n9 to n0
28         el09 from n9 to n8
29         el10 from n8 to n6
30         el11 from n5 to n7
31       group hcfgroup cfactor 4.0 function roadFunction members
32         eh01 from n5 to n3
33         eh02 from n3 to n0
34     sources
35       // code to be added
36   agentTypes
37     agentType myDeliveryType congestionFactor 0 maxWeight 180
38       behaviour awt awaitTour when finished do die
39       behaviour die vanish
```

Task: Q02ATAG (2/2)

Task (continuation)

Snippet1:

```
35 n1 isDepot stuff sprouts myDeliveryType customers n0, n2, n4, n5, n6 at 0
```

Snippet 2:

```
35 n0 isDepot stuff sprouts myDeliveryType customers n0, n2, n4, n5, n6 at 0
```

Snippet 3:

```
35 n0 isDepot stuff sprouts myDeliveryType customers n1, n2, n3, n5, n6 at 0
```

Snippet 4:

```
35 n0 isDepot stuff sprouts myDeliveryType customers n1, n2, n4, n5, n6 at 0
```

Correct solution

☑Snippet 1, ☑Snippet 2, ☑Snippet 3

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q03ATALL (1/4)

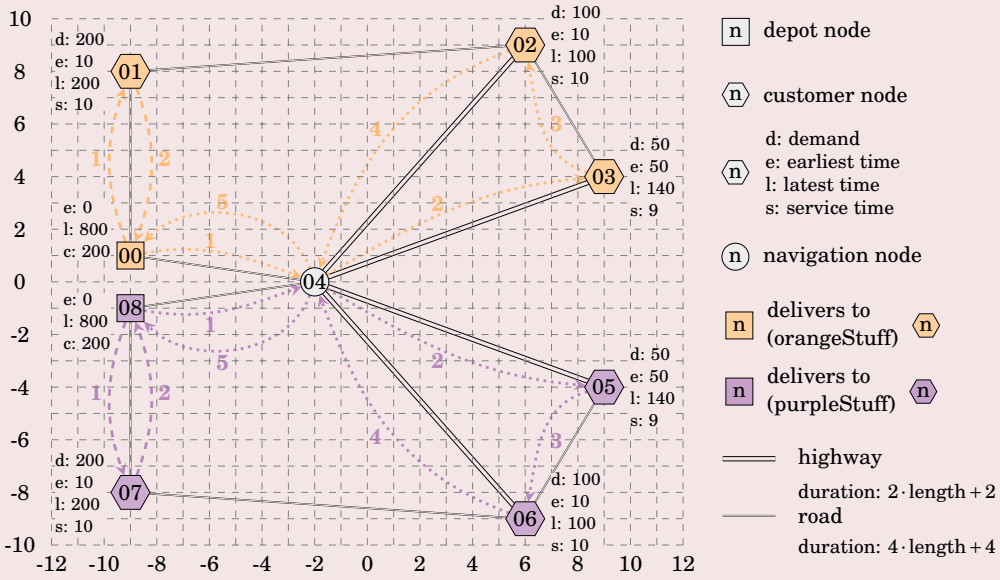
Introduction

In this task you see the illustration of a Network (comprised of customers, demands, roads / highways etc.). In addition, the illustration also shows optised vehicle routes for a VRP based on the illustrated network.Your task is to determine which of the three models / programs corresponds to the illustrated network.

Task

Q1 – Which of the three programs corresponds to the illustration?

○Program 1	○Program 2	○Program 3



Task: Q03ATALL (2/4)

Task (continuation)

Program 1

```
model q11AllStartState
products
  orngStuff weight 1.0
  prplStuff weight 1.0
functions
  durationFunction roadFunction  4 * length + 4
  durationFunction highwayFunction  2 * length + 2
network
  nodes
    n0 at (-9, 1) isDepot orngStuff sprouts orngVehis customers n1, n2, n3 at 0 latestTime 800
    n1 at (-9, 8) hasDemand orngStuff units 200 earliestTime 10 latestTime 200 serviceTime 10
    n2 at (6, 9) hasDemand orngStuff units 100 earliestTime 10 latestTime 100 serviceTime 10
    n3 at (9, 4) hasDemand orngStuff units 50 earliestTime 50 latestTime 140 serviceTime 9
    n4 at (-2, 0)
    n5 at (9, -4) hasDemand prplStuff units 50 earliestTime 50 latestTime 140 serviceTime 9
    n6 at (6, -9) hasDemand prplStuff units 100 earliestTime 10 latestTime 100 serviceTime 10
    n7 at (-9, -8) hasDemand prplStuff units 200 earliestTime 10 latestTime 200 serviceTime 10
    n8 at (-9, -1) isDepot prplStuff sprouts prplVehis customers n7, n6, n5 at 0 latestTime 800
  edges
    group roadGroup function roadFunction members
      h1 from n2 to n4
      h2 from n3 to n4
      h3 from n6 to n4
      h4 from n5 to n4
    group highwayGroup function highwayFunction members
      r1 from n0 to n1
      r2 from n0 to n4
      r3 from n1 to n2
      r4 from n2 to n3
      r5 from n8 to n7
      r6 from n8 to n4
      r7 from n7 to n6
      r8 from n5 to n6
agentTypes
  agentType orngVehis congestionFactor 0 maxWeight 200
    behaviour awt awaitTour when finished do die
    behaviour die vanish
  agentType prplVehis congestionFactor 0 maxWeight 200
    behaviour awt awaitTour when finished do die
    behaviour die vanish
```

Task: Q03ATALL (3/4)

Task (continuation)

Program 2

```
model q11AllStartState
products
  orngStuff weight 1.0
  prplStuff weight 1.0
functions
  durationFunction roadFunction  4 * length + 4
  durationFunction highwayFunction  2 * length + 2
network
  nodes
    n0 at (-9, 1) isDepot orngStuff sprouts orngVehis customers n1, n2, n3 at 0 latestTime 800
    n1 at (-9, 8) hasDemand orngStuff units 200 earliestTime 10 latestTime 200 serviceTime 10
    n2 at (6, 9) hasDemand orngStuff units 100 earliestTime 10 latestTime 100 serviceTime 10
    n3 at (9, 4) hasDemand orngStuff units 50 earliestTime 50 latestTime 140 serviceTime 9
    n4 at (-2, 0)
    n5 at (9, -4) hasDemand prplStuff units 50 earliestTime 50 latestTime 140 serviceTime 9
    n6 at (6, -9) hasDemand prplStuff units 100 earliestTime 10 latestTime 100 serviceTime 10
    n7 at (-9, -8) hasDemand prplStuff units 200 earliestTime 10 latestTime 200 serviceTime 10
    n8 at (-9, -1)isDepot prplStuff sprouts prplVehis customers n7, n6, n5 at 0 latestTime 800
  edges
    group roadGroup function roadFunction members
      r1 from n0 to n1
      r2 from n0 to n4
      r3 from n1 to n2
      r4 from n2 to n3
      r5 from n8 to n7
      r6 from n8 to n4
      r7 from n7 to n6
      r8 from n5 to n6
    group highwayGroup function highwayFunction members
      h1 from n2 to n4
      h2 from n3 to n4
      h3 from n6 to n4
      h4 from n5 to n4
agentTypes
  agentType orngVehis congestionFactor 0 maxWeight 200
    behaviour awt awaitTour when finished do die
    behaviour die vanish
  agentType prplVehis congestionFactor 0 maxWeight 200
    behaviour awt awaitTour when finished do die
    behaviour die vanish
```

Task: Q03ATALL (4/4)

Task (continuation)

Program 2

```
model q11AllStartState
products
  orngStuff weight 1.0
  prplStuff weight 1.0
functions
  durationFunction roadFunction  4 * length + 4
  durationFunction highwayFunction  2 * length + 2
network
  nodes
    n0 at (-9, 1) isDepot prplStuff sprouts prplVehis customers n7, n6, n5 at 0 latestTime 800
    n1 at (-9, 8) hasDemand orngStuff units 200 earliestTime 10 latestTime 200 serviceTime 10
    n2 at (6, 9) hasDemand orngStuff units 100 earliestTime 10 latestTime 100 serviceTime 10
    n3 at (9, 4) hasDemand orngStuff units 50 earliestTime 50 latestTime 140 serviceTime 9
    n4 at (-2, 0)
    n5 at (9, -4) hasDemand prplStuff units 50 earliestTime 50 latestTime 140 serviceTime 9
    n6 at (6, -9) hasDemand prplStuff units 100 earliestTime 10 latestTime 100 serviceTime 10
    n7 at (-9, -8) hasDemand prplStuff units 200 earliestTime 10 latestTime 200 serviceTime 10
    n8 at (-9, -1) isDepot orngStuff sprouts orngVehis customers n1, n2, n3 at 0 latestTime 800
  edges
    group roadGroup function roadFunction members
      r1 from n0 to n1
      r2 from n0 to n4
      r3 from n1 to n2
      r4 from n2 to n3
      r5 from n8 to n7
      r6 from n8 to n4
      r7 from n7 to n6
      r8 from n5 to n6
    group highwayGroup function highwayFunction members
      h1 from n2 to n4
      h2 from n3 to n4
      h3 from n6 to n4
      h4 from n5 to n4
agentTypes
  agentType orngVehis congestionFactor 0 maxWeight 200
    behaviour awt awaitTour when finished do die
    behaviour die vanish
  agentType prplVehis congestionFactor 0 maxWeight 200
    behaviour awt awaitTour when finished do die
    behaviour die vanish
```

Correct solution

◉ Program 2

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q04ATNW (1/3)

Introduction

In this task you are shown a program and four different graphical networks. One of these networks is exactly described (modelled) by the program. For the other three networks the program is not completely right. It is your task to find and report the exactly modelled network.

Task

Which of the above networks results from the given Athos model?

○Network A	○Network B	○Network C	○Network D

```
1  model q04atnw
2  products
3    stuff weight 1.0
4  functions
5    durationFunction highwayFunction  1.5 * length
6    durationFunction roadFunction  4 * length + 5
7  network
8    nodes
9      n0 at (0, -6) isDepot stuff sprouts vhcls customers n1, n2, n3, n5, n6 at 0 latestTime 500
10     n1 at (-9, 4) hasDemand stuff units 15 earliestTime 15 latestTime 120 serviceTime 5
11     n2 at (7, -9) hasDemand stuff units 20 earliestTime 10 latestTime 130 serviceTime 7
12     n3 at (8, 5)  hasDemand stuff units 50 earliestTime 20 latestTime 90  serviceTime 10
13     n4 at (2, 0)
14     n5 at (-2, -1) hasDemand stuff units 25 earliestTime 90 latestTime 250 serviceTime 10
15     n6 at (-8, -6) hasDemand stuff units 25 earliestTime 90 latestTime 270 serviceTime 5
16   edges
17     group roadGroup function roadFunction members
18       road1 from n0 to n5
19       road2 from n0 to n4
20       road3 from n5 to n4
21       road4 from n4 to n3
22       road5 from n5 to n1
23       road6 from n0 to n6
24     group highwayGroup function roadFunction members
25       highway1 from n1 to n3
26       highway2 from n3 to n2
27       highway3 from n2 to n6
28       highway4 from n6 to n1
29 agentTypes
30   agentType vhcls congestionFactor 0 maxWeight 180
31     behaviour awt awaitTour when finished do die
32     behaviour die vanish
```

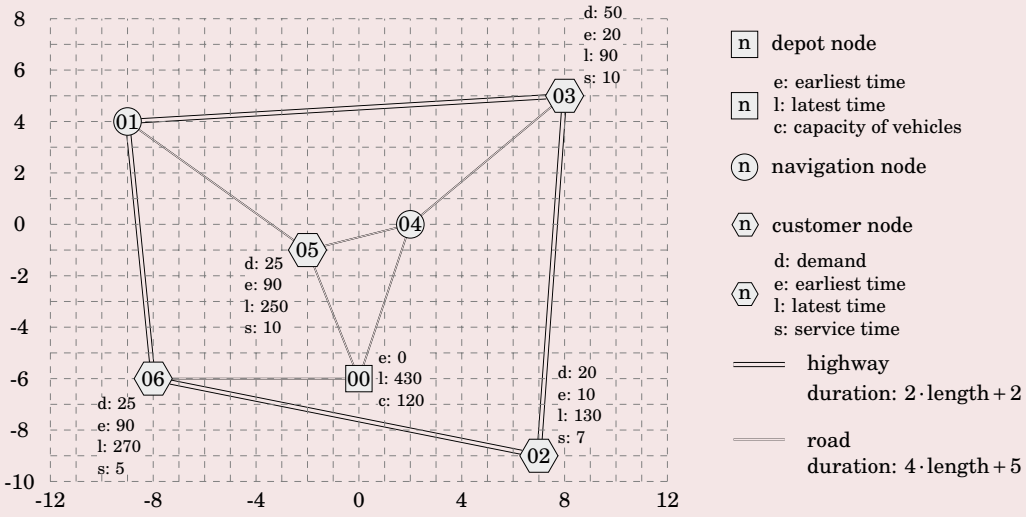
Task: Q04ATNW (2/3)

Introduction

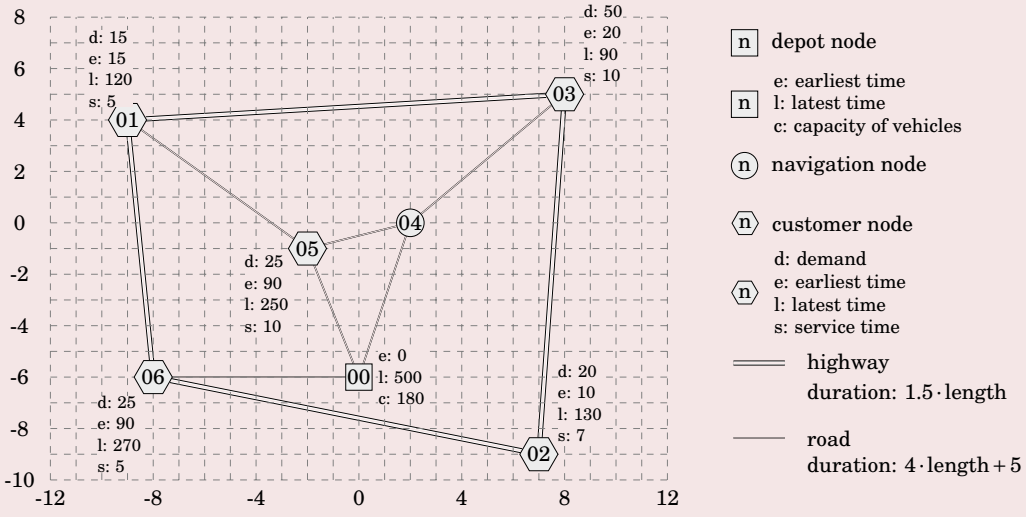
In this task you are shown a program and four different graphical networks. One of these networks is exactly described (modelled) by the program. For the other three networks the program is not completely right. It is your task to find and report the exactly modelled network.

Task (continuation)

Network A



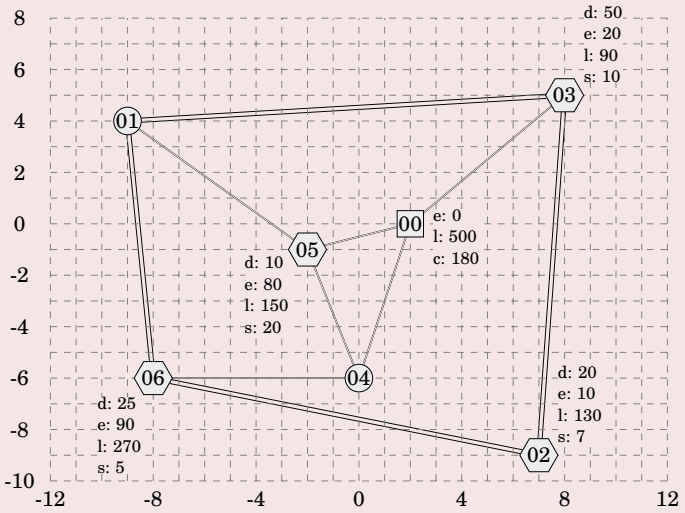
Network B



Task: Q04ATNW (3/3)

Task (continuation)

Network C



n depot node

n e: earliest time
l: latest time
c: capacity of vehicles

n navigation node

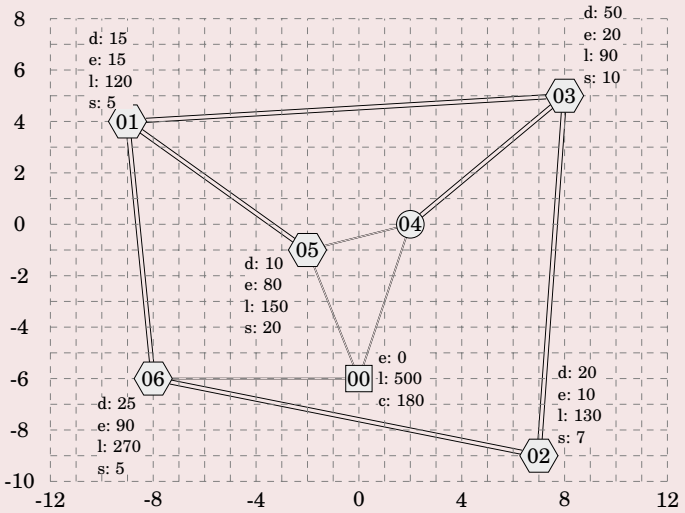
n customer node

n d: demand
e: earliest time
l: latest time
s: service time

highway
duration: $1.5 \cdot \text{length}$

road
duration: $4 \cdot \text{length} + 5$

Network D



n depot node

n e: earliest time
l: latest time
c: capacity of vehicles

n navigation node

n customer node

n d: demand
e: earliest time
l: latest time
s: service time

highway
duration: $1.5 \cdot \text{length}$

road
duration: $3 \cdot \text{length} + 3$

Correct solution

◉ Network B

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q04ATAG (1/3)

Introduction

In this task you are shown a program and four graphical networks on which four different tours are depicted. One of these networks shows a tour that is exactly described (modelled) by the program. For the other three tours the program is not completely right. It is your task to find and report the exactly modelled tour.

NOTE: If a tour step connects two nodes that do not share an edge, this means that the actual path from the start node to the target node of the respective step is not important. However, only nodes of the drawn tour are serviced!

Task

Which of the presented tours for a vehicle may result from the program given below?

○Tour 1	○Tour 2	○Tour 3	○Tour 4

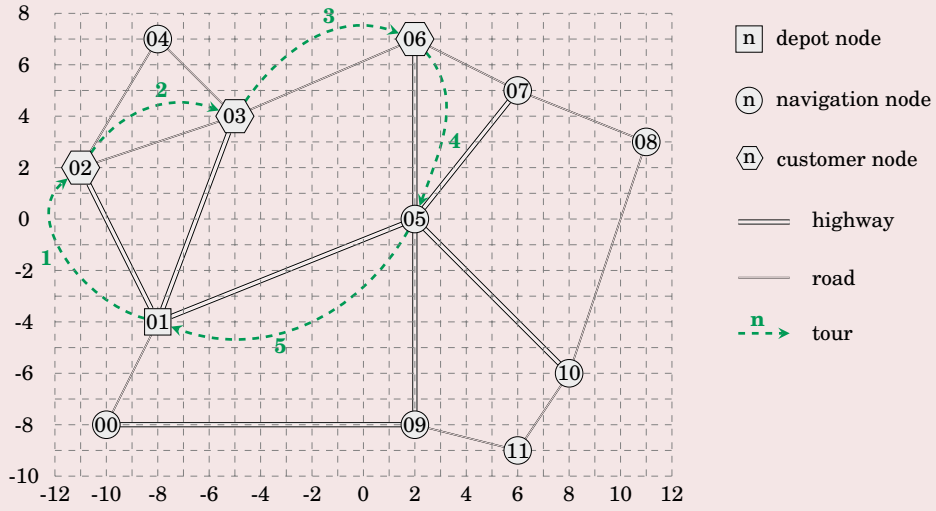
```
functions
  durationFunction highwayFunction  length
  durationFunction roadFunction  3 * length + 5
network
  nodes
    n0 at (-10, -8) isDepot stuff sprouts vehicles customers n2, n3, n5, n6 at 0 latestTime 500
    n1 at (-8, -4)
    n2 at (-11, 2) hasDemand stuff units 20 earliestTime 10 latestTime 130 serviceTime 7
    n3 at (-5, 4)  hasDemand stuff units 50 earliestTime 20 latestTime 90  serviceTime 10
    n4 at (-8, 7)
    n5 at (2, 0) hasDemand stuff units 25 earliestTime 90 latestTime 250 serviceTime 10
    n6 at (2, 7) hasDemand stuff units 25 earliestTime 90 latestTime 250 serviceTime 10
    n7 at (6, 5)
    n8 at (11, 3)
    n9 at (2, -8)
    n10 at (8, -6)
    n11 at (6, -9)
  edges
    group roadGroup function roadFunction members
      road1 from n0 to n1
      road2 from n2 to n4
      road3 from n2 to n3
      road4 from n4 to n3
      road5 from n3 to n6
      road6 from n6 to n7
      road7 from n7 to n8
      road8 from n8 to n10
      road9 from n10 to n11
      road10 from n11 to n9
    group highwayGroup function highwayFunction members
      highway1 from n1 to n2
      highway2 from n1 to n3
      highway3 from n1 to n5
      highway4 from n5 to n6
      highway5 from n5 to n7
      highway6 from n5 to n10
      highway7 from n5 to n9
      highway8 from n9 to n0
agentTypes
  agentType vehicles maxWeight 180
    behaviour awt awaitTour when finished do die
    behaviour die vanish
```

Task: Q04ATAG (2/3)

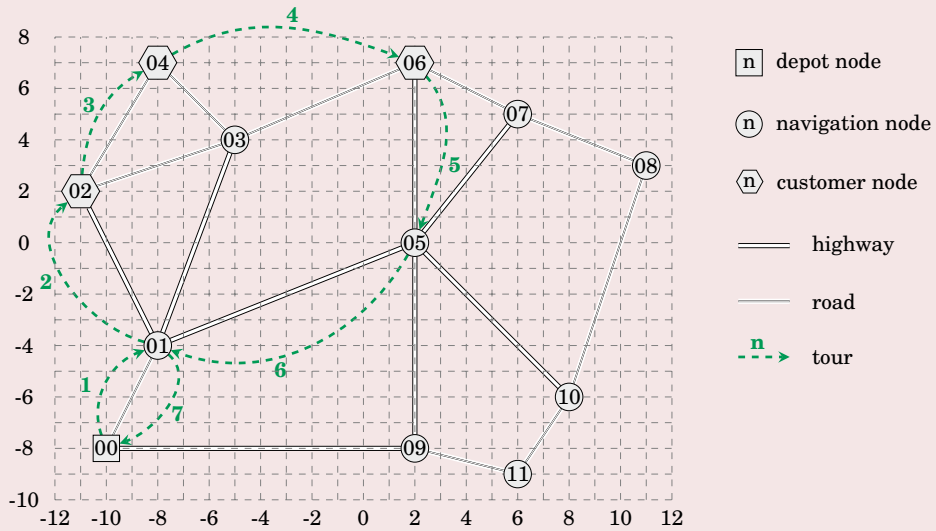
Introduction

Task (continuation)

Tour 1



Tour 2

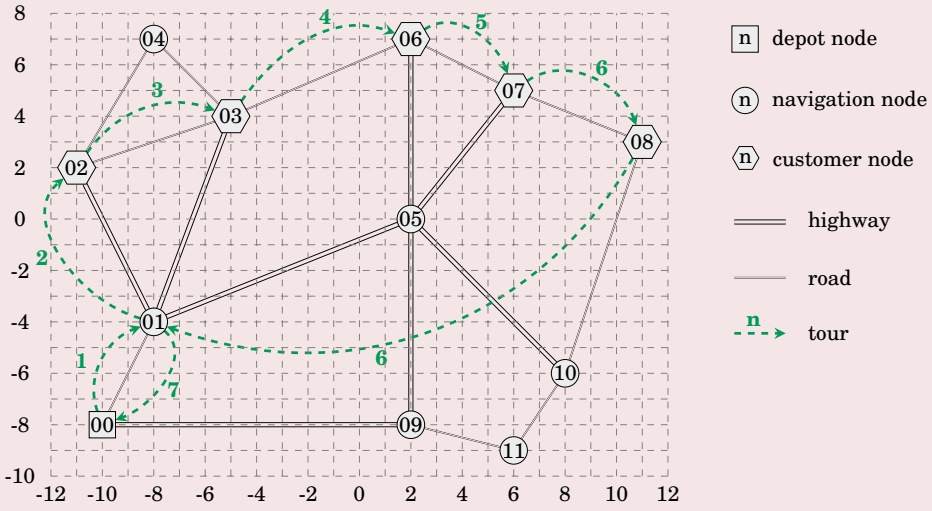


Task: Q04ATAG (3/3)

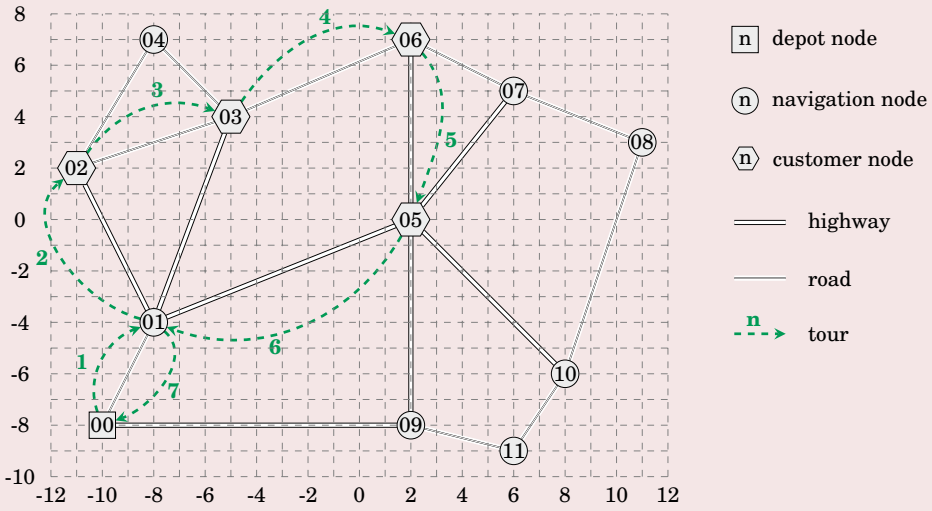
Introduction

Task (continuation)

Tour 3



Tour 4



Correct solution

☑Tour 4

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q05ATNW (1/2)

Introduction

In this task you will find a program and two networks. The first of the two networks is described by the program. If some of the lines of the program are changed, the program is a description of the second network. It is your task to find these lines and report them.

Task

The Athos model below represents network 1 . What lines of the Athos model would have to be changed if you wanted to model network 2 (multiple answers possible)?

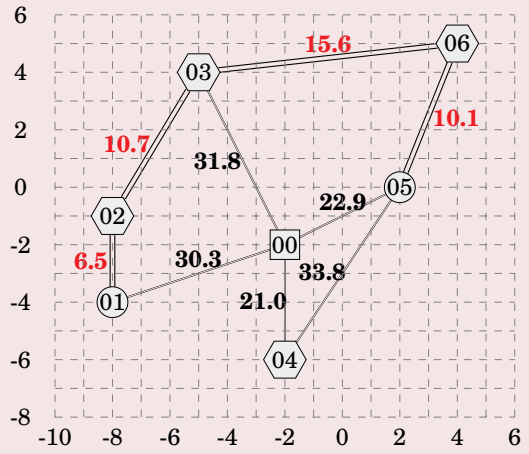
☐ Line 1	☐ Line 2	☐ Line 3	☐ Line 4	☐ Line 5	☐ Line 6
☐ Line 7	☐ Line 8	☐ Line 9	☐ Line 10	☐ Line 11	☐ Line 12
☐ Line 13	☐ Line 14	☐ Line 15	☐ Line 16	☐ Line 17	☐ Line 18
☐ Line 19	☐ Line 20	☐ Line 21	☐ Line 22	☐ Line 23	☐ Line 24
☐ Line 25	☐ Line 26	☐ Line 27	☐ Line 28	☐ Line 29	☐ Line 30
☐ Line 31	☐ Line 32	☐ Line 33	☐ Line 34	☐ Line 35	☐ Line 36

```
1  model q5nw
2    products stuff weight 1.0
3    functions
4      durationFunction roadFunction  4 * length + 5
5      durationFunction highwayFunction  1.5 * length + 2
6    network
7      nodes
8        n0 at (-2, -2) isDepot stuff sprouts vehicles customers n2, n3, n4, n6 at 0 latestTime 500
9        n1 at (-8, -4)
10       n2 at (-8, -1) hasDemand stuff units 20 earliestTime 10 latestTime 130 serviceTime 7
11       n3 at (-5,  4) hasDemand stuff units 50 earliestTime 20 latestTime 90  serviceTime 10
12       n4 at (-2, -6) hasDemand stuff units 25 earliestTime 90 latestTime 250 serviceTime 10
13       n5 at (2,  0)
14       n6 at (4,  5) hasDemand stuff units 25 earliestTime 90 latestTime 250 serviceTime 10
15     edges
16       group roadGroup function roadFunction
17         members
18           road1 from n0 to n4
19           road2 from n0 to n5
20           road3 from n0 to n1
21           road4 from n0 to n3
22           road5 from n4 to n5
23       group highwayGroup function highwayFunction
24         members
25           highway1 from n1 to n2
26           highway2 from n2 to n3
27           highway3 from n3 to n6
28           highway4 from n6 to n5
```

Task: Q05ATNW (2/2)

Task (continuation)

Network 1 (currently modelled)



n depot node

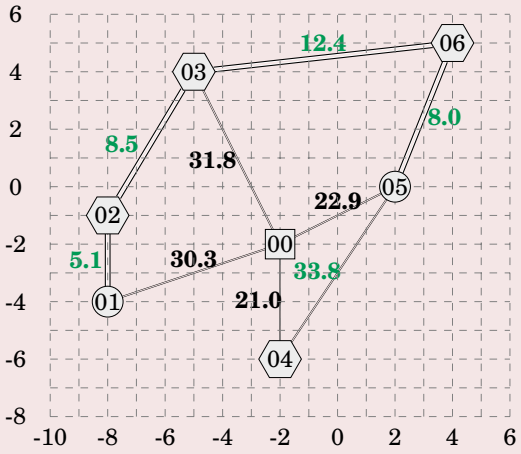
(n) navigation node

⟨n⟩ customer node

d highway
duration: 1.5 · length + 2

d road
duration: 4 · length + 5

Network 2 (target state)



n depot node

(n) navigation node

⟨n⟩ customer node

d highway
duration: 1.2 · length + 1.5

d road
duration: 4 · length + 5

Correct solution

☑Line 5

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q05ATAG (1/2)

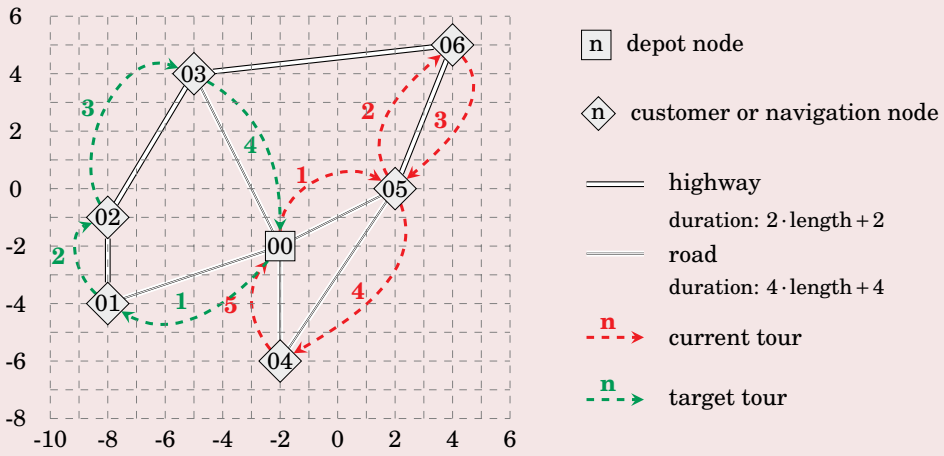
Introduction

In this task, you will find the graphical representation of a network together with a program. The network shows two tours. The program describes the network and a problem for which one of the tours is likely to be optimal. By changing some lines of the program, the program describes a problem for which the other tour is likely to be optimal. It is your task to find and report these lines.

Task

In the picture below, the red tour is a likely result from the problem modelled in the Athos code given at the bottom. What lines of the Athos model have to be changed so that the green tour is likely to result from the model?

☐ Line 1	☐ Line 2	☐ Line 3	☐ Line 4	☐ Line 5	☐ Line 6
☐ Line 7	☐ Line 8	☐ Line 9	☐ Line 10	☐ Line 11	☐ Line 12
☐ Line 13	☐ Line 14	☐ Line 15	☐ Line 16	☐ Line 17	☐ Line 18
☐ Line 19	☐ Line 20	☐ Line 21	☐ Line 22	☐ Line 23	☐ Line 24
☐ Line 25	☐ Line 26	☐ Line 27	☐ Line 28	☐ Line 29	☐ Line 30
☐ Line 31					



Task: Q05ATAG (2/2)

Task (continuation)

```
1  model q5ag
2    products
3      stuff weight 1.0
4    functions
5      durationFunction roadFunction  4 * length + 4
6      durationFunction highwayFunction  2 * length + 2
7    network
8      nodes
9        n0 at (-2, -2) isDepot stuff sprouts vehicles customers n4, n5, n6 at 0 latestTime 20000
10       n1 at (-8, -4) hasDemand stuff units 20 earliestTime 0 latestTime 20000 serviceTime 7
11       n2 at (-8, -1) hasDemand stuff units 20 earliestTime 0 latestTime 20000 serviceTime 7
12       n3 at (-5,  4) hasDemand stuff units 50 earliestTime 0 latestTime 20000  serviceTime 10
13       n4 at (-2, -6) hasDemand stuff units 25 earliestTime 0 latestTime 20000 serviceTime 10
14       n5 at (2,  0) hasDemand stuff units 25 earliestTime 0 latestTime 20000 serviceTime 10
15       n6 at (4,  5) hasDemand stuff units 25 earliestTime 0 latestTime 20000 serviceTime 10
16     edges
17       group roadGroup function roadFunction members
18         road1 from n0 to n4
19         road2 from n0 to n5
20         road3 from n0 to n1
21         road4 from n0 to n3
22         road5 from n4 to n5
23       group highwayGroup function highwayFunction members
24         highway1 from n1 to n2
25         highway2 from n2 to n3
26         highway3 from n3 to n6
27         highway4 from n6 to n5
28   agentTypes
29     agentType vehicles congestionFactor 0 maxWeight 180
30       behaviour awt awaitTour when finished do die
31       behaviour die vanish
```

Correct solution

☑Line 9

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q06ATNW (1/4)

Introduction

In this task, you see a program that describes a network for a VRPTW. In addition, you see four more programs. Some of these programs describe the exact same network as the first program, even though they syntactically deviate from the first program. It is your task to find and report these equivalent programs.

Task

Look at the "Program to match" below. One or more of the four possible matches produce an equivalent network. Tick the respective boxes!

☐ Possible match 01	☐ Possible match 02	☐ Possible match 03	☐ Possible match 04

Program to match

```
functions
  durationFunction roadFunction  2 * length + 4
  durationFunction highwayFunction  4 * length + 2
  durationFunction specialFunction 3 * length + 3
network
  nodes
    n0 at (-2, -2)
    n1 at (-8, -4)
    n2 at (-8, -1)
    n3 at (-5,  4)
    n4 at (-2, -6)
    n5 at (2,  0)
    n6 at (4,  5)
  edges
    group roadGroup function roadFunction members
      road1 from n0 to n4
      road2 from n0 to n5 function specialFunction
      road3 from n0 to n1
      road4 from n0 to n3
      road5 from n4 to n5
    group highwayGroup function highwayFunction members
      highway1 from n1 to n2
      highway2 from n2 to n3
      highway3 from n3 to n6 function specialFunction
      highway4 from n6 to n5
```

Task: Q06ATNW (2/4)

Task (continuation)

Matching option 1

```
functions
  durationFunction roadFunction  2 * length + 4
  durationFunction highwayFunction  4 * length + 2
  durationFunction specialFunction 3 * length + 3
network
  nodes
    n0 at (-2, -2)
    n1 at (-8, -4)
    n2 at (-8, -1)
    n3 at (-5,  4)
    n4 at (-2, -6)
    n5 at (2,  0)
    n6 at (4,  5)
  edges
    group roadGroup function roadFunction members
      road1 from n0 to n4
      road2 from n0 to n5
      road3 from n0 to n1
      road4 from n0 to n3
      road5 from n4 to n5 function specialFunction
    group highwayGroup function highwayFunction members
      highway1 from n1 to n2
      highway2 from n2 to n3 function specialFunction
      highway3 from n3 to n6
      highway4 from n6 to n5
```

Matching option 2

```
functions
  durationFunction roadFunction  2 * length + 4
  durationFunction highwayFunction  4 * length + 2
  durationFunction specialFunction 3 * length + 3
  network
    nodes
      n0 at (-2, -2)
      n1 at (-8, -4)
      n2 at (-8, -1)
      n3 at (-5,  4)
      n4 at (-2, -6)
      n5 at (2,  0)
      n6 at (4,  5)
    edges
      group roadGroup function roadFunction members
        road1 from n0 to n4
        road3 from n0 to n1
        road4 from n0 to n3
        road5 from n4 to n5
      group highwayGroup function highwayFunction members
        highway1 from n1 to n2
        highway2 from n2 to n3
        highway4 from n6 to n5
      group fastwayGroup function specialFunction members
        fastway1 from n0 to n5
        fastway2 from n3 to n6
```

Task: Q06ATNW (3/4)

Task (continuation)

Matching option 3

```
functions
  durationFunction roadFunction  2 * length + 4
  durationFunction highwayFunction  4 * length + 2
  durationFunction specialFunction 3 * length + 3
network
  nodes
    n0 at (-2, -2)
    n1 at (-8, -4)
    n2 at (-8, -1)
    n3 at (-5,  4)
    n4 at (-2, -6)
    n5 at (2,  0)
    n6 at (4,  5)
  edges
    group roadGroup function roadFunction members
      road1 from n0 to n4
      road3 from n0 to n1
      road4 from n0 to n3
      road5 from n4 to n5
    group highwayGroup function highwayFunction members
      highway1 from n1 to n2
      highway3 from n3 to n6
      highway4 from n6 to n5
    group fastwayGroup function specialFunction members
      fastway2 from n0 to n5  function highwayFunction
      fastway1 from n2 to n3 function specialFunction
```

Matching option 4

```
functions
  durationFunction roadFunction  2 * length + 4
  durationFunction highwayFunction  4 * length + 2
  durationFunction specialFunction 3 * length + 3
network
  nodes
    n0 at (-2, -2)
    n1 at (-8, -4)
    n2 at (-8, -1)
    n3 at (-5,  4)
    n4 at (-2, -6)
    n5 at (2,  0)
    n6 at (4,  5)
  edges
    road1 from n0 to n4 function roadFunction
    road2 from n0 to n5  function specialFunction
    road3 from n0 to n1 function roadFunction
    road4 from n0 to n3 function roadFunction
    road5 from n4 to n5 function roadFunction
    highway1 from n1 to n2 function highwayFunction
    highway2 from n2 to n3 function highwayFunction
    highway3 from n3 to n6 function specialFunction
    highway4 from n6 to n5 function highwayFunction
```

Task: Q06ATNW (4/4)

Task (continuation)

Correct solution

☑Possible match 2, Possible match 4

Evaluation 2020

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	1 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Evaluation 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	1 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q07ATALL (1/2)

Introduction

In this task, you first see a complete program. After that, you are shown excerpts from this program and you are asked to associate the correct semantics (meaning) with the language elements shown in the excerpt. Thus, it is your task to associate the correct semantics to pre-selected language elements.

Task

```
1  model q6nw
2  products
3    stuff weight 1.0
4  functions
5    durationFunction roadFunction  2.5 * length + 3.2
6    durationFunction highwayFunction  1.2 * length + 1.3
7  network
8    nodes
9      n0 at (-2, -2) isDepot stuff sprouts vehicles customers n1, n2, n4, n5 at 0 latestTime 850
10     n1 at (3, -4) hasDemand stuff units 20 earliestTime 10 latestTime 300 serviceTime 7
11     n2 at (0, -3) hasDemand stuff units 20 earliestTime 20 latestTime 250 serviceTime 7
12     n3 at (-5,  4) hasDemand stuff units 50 earliestTime 0 latestTime 140  serviceTime 10
13     n4 at (-2, -6) hasDemand stuff units 25 earliestTime 0 latestTime 120 serviceTime 10
14     n5 at (2,  0) hasDemand stuff units 25 earliestTime 0 latestTime 140 serviceTime 10
15   edges
16     group roadGroup function roadFunction members
17       road1 from n0 to n4
18       road2 from n0 to n5
19       road3 from n0 to n2
20       road4 from n0 to n3
21       road5 from n1 to n2
22       road6 from n1 to n4
23     group highwayGroup function highwayFunction members
24       highway3 from n3 to n5
25       highway4 from n2 to n5
26   agentTypes
27     agentType vehicles congestionFactor 0 maxWeight 180
28       behaviour awt awaitTour when finished do die
29       behaviour die vanish
```

1st Element

```
4  functions
5    durationFunction roadFunction  2.5 * length + 3.2
```

Which description of the sematics of the the durationFunction element is the most appropriate?

- ○ It is used for single customers (Services). Together with an expression that refers to the respective customer it determines how long the time window of this customer is opened.
- ○ It is used for single vehicles (agents). Together with an expression it determines the costs that occur upon deployment of the respective vehicle (agent).
- ○ It is used for single edges, that is the connection between two customers. Together with an expression it determines how long it takes a vehicle (agent) to travel the respective edge.
- ○ It is used for single depots. Together with an expression it determines the capacity of the vehicles (agents) starting from that depot.

Task: Q07ATALL (2/2)

Task (continuation)

2nd Element

```
15 edges
16   group roadGroup function roadFunction members
```

Which explanation concerning the meaning of the depicted group element in combination with the ensuing definition of edges in the context of the complete program is most appropriate?

- ○ It allows to group agents (vehicles) that will then jointly travel the respective edge
- ○ All edges (connections between two nodes) must be associated with a group because only via a group it is possible to set the duration function of an edge.
- ○ Edges of a given group automatically constitute a path and thus share the same congestionFactor.
- ○ They are an optional language element that may facillitate the definition of edges. Especially in cases in which a large number of edges are assigned the same (duration) function.

Correct solution

1st Element

- ◉ It is used for single edges, that is the connection between two customers. Together with an expression it determines how long it takes a vehicle (agent) to travel the respective edge.

2nd Element

- ◉ They are an optional language element that may facillitate the definition of edges. Especially in cases in which a large number of edges are assigned the same (duration) function.

Evaluation 2020, 2021

Non-attempt iff both tasks were not answered.

	Parameter	Value	Parameter	Value
1st Element	Answer Cell	Yes	NegPointsP	5 (default)
	CorAnswersString	Yes	WrongAnswAllowed	0 (default)
	Points	5	MissingAnswAllowed	0 (default)
	WrgAnswersString	No (default)	MaxPointsToGet	5 (default)
	Parameter	Value	Parameter	Value
2nd Element	Answer Cell	Yes	NegPointsP	5 (default)
	CorAnswersString	Yes	WrongAnswAllowed	0 (default)
	Points	5	MissingAnswAllowed	0 (default)
	WrgAnswersString	No (default)	MaxPointsToGet	5 (default)

Task: Q08ATALL (1/2)

Introduction

In this task, you first see a complete program. The program features some comments. These comments represent TODOs, i.e. future programming tasks. In addition, you'll find some questions that ask for the elements, that will be affected by these TODOs. Your task is to answer these questions correctly.[a]

Task

```
01 model q08all
02 products
03   sp weight 1.0
04   pp weight 1.0
05 functions
06   durationFunction roadFunction  2.5 * length + 3.2
07   durationFunction highwayFunction  1.2 * length + 1.3
08 network
09   // TODO: The customer with the longest service time must be added as a customer to the
10   // respective depot
11  nodes
12   n0 at (2,4) isDepot sp sprouts vehicles customers n1, n4, n6 at 0 latestTime 850
13   n1 at (3,-4) hasDemand sp units 20 earliestTime 10 latestTime 300 serviceTime 7
14   n2 at (0,-3) hasDemand sp units 20, pp units 10 earliestTime 20 latestTime 250 serviceTime 7
15   n3 at (-5,4) hasDemand sp units 50 earliestTime 0 latestTime 140  serviceTime 30
16   n4 at (-2,-6) hasDemand sp units 25 earliestTime 0 latestTime 120 serviceTime 10
17   n5 at (2,0) hasDemand sp units 30, pp units 40  earliestTime 0 latestTime 140 serviceTime 10
18   n6 at (-2,-2) hasDemand sp units 60, pp units 70 earliestTime 0 latestTime 150 serviceTime 10
19   n7 at (-5,-2) hasDemand pp units 20 earliestTime 20 latestTime 220 serviceTime 8
20   n8 at (-5,-6) isDepot pp sprouts vehicles customers n5, n6, n7 at 0 latestTime 500
21  edges
22   group roadGroup function roadFunction members
23     road01 from n0 to n4
24     road02 from n0 to n5
25     road03 from n3 to n6
26     road04 from n0 to n3
27     road05 from n1 to n2
28     road06 from n1 to n4
29     road07 from n3 to n0
30     road08 from n5 to n0
31     road09 from n3 to n7
32     road10 from n7 to n8
33     road11 from n6 to n2
34     road12 from n8 to n4
35   group highwayGroup function highwayFunction members
36     // TODO: two more highways will later be modelled here:
37     // one going from the soap depot to the customer with the highes demands in both soap
38     // and paper. The other going from the customer with the highest soap and
39     // paper demands to the paper depot.
40     highway3 from n3 to n5
41     highway4 from n2 to n5
42 agentTypes
43   agentType vehicles congestionFactor 0 maxWeight 180
44   behaviour awt awaitTour when finished do die
45   behaviour die vanish
```

1. Which customer must be added?

Read the comment in lines 09 and 10. Which customer must be added to the respective depot according to the comment?

Note: Some of the provided answers might not be customers.

○ n0	○ n1	○ n2
○ n3	○ n4	○ n5
○ n6	○ n7	○ n8

[a] Coordinates were originally specified with a blank space after the comma. Original product names were 'soap' and 'paper'.

Task: Q08ATALL (2/2)

Task (continuation)

2. Which is the correct depot?
Read the comment in lines 09 and 10. To which depot must the customer be added to?
Note: Some of the provided answers might not be depots.

○ n0	○ n1	○ n2
○ n3	○ n4	○ n5
○ n6	○ n7	○ n8

3. Which nodes are affected?
Read the comment that spans from lines 36 to line 39. What are the three nodes this comment refers to?
Note: Some of the provided answers might not be customers.

☐ n0	☐ n1	☐ n2
☐ n3	☐ n4	☐ n5
☐ n6	☐ n7	☐ n8

Correct solution

1.	◉ n3
2.	◉ n0
3.	☑ n0, ☑ n6, ☑ n8

Evaluation 2020, 2021

Non-attempt iff all three tasks were not answered.

1.

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	2
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	2	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	2

2.

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	2
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	2	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	2

3.

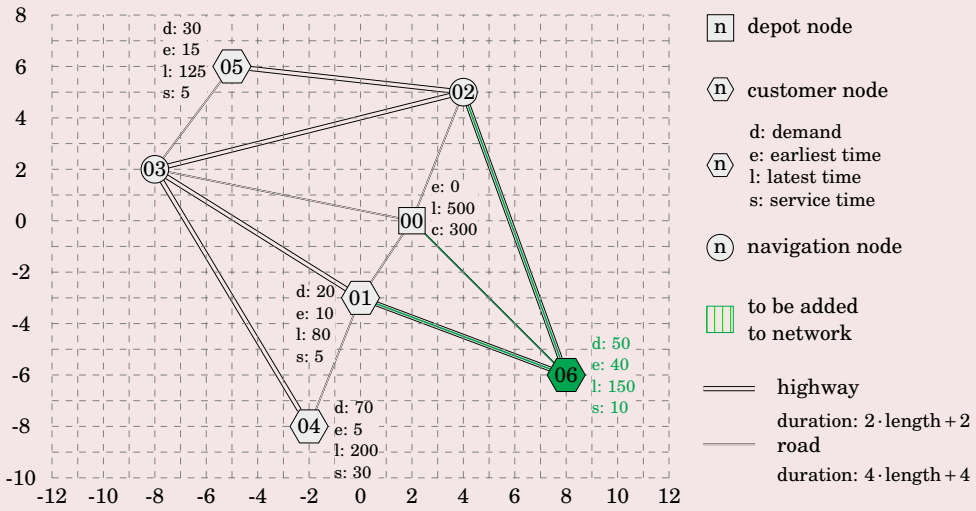
Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	2
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	2	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	6

Task: Q09ATNW (1/2)

Introduction

In this task you'll find a graphical representation of a network. The network comprises a set of customers with demands, time windows and service times. Further below is the corresponding program. However, the program still shows some gaps. The gaps can also be seen in the illustration: they correspond to the elements drawn using green color. Your task is to fill in the gaps so that the program describes the complete network.

Task




```
1  model q9nw
2  products
3    stuff weight 1.0
4  functions
5    durationFunction roadFunction  4 * length + 4
6    durationFunction highwayFunction  2 * length + 2
7  network
8    nodes
9      n0 at (2, 0) isDepot stuff sprouts vehicles customers n1, n4, n5 at 0 latestTime 500
10     n1 at (0, -3) hasDemand stuff units 20 earliestTime 10 latestTime 80 serviceTime 5
11     n2 at (4, 5)
12     n3 at (-8, 2)
13     n4 at (-2, -8) hasDemand stuff units 70 earliestTime 5 latestTime 200 serviceTime 30
14     n5 at (-5, 6) hasDemand stuff units 30 earliestTime 15 latestTime 125 serviceTime 5
15     // TASK 1: SOME TEXT TO BE ADDED HERE
16   edges
17     group roadGroup function roadFunction members
18       road1 from n0 to n1
19       road2 from n0 to n2
20       road3 from n0 to n3
21       road4 from n1 to n4
22       road5 from n3 to n5
23       // TASK 2: SOME TEXT TO BE ADDED HERE
24     group highwayGroup [type1 ultraThin green] function highwayFunction members
25       highway1 from n5 to n2
26       highway2 from n3 to n2
27       highway3 from n3 to n1
28       highway4 from n3 to n4
29       // TASK 3: SOME TEXT TO BE ADDED HERE
30 agentTypes
31   agentType vehicles congestionFactor 0 maxWeight 180
32     behaviour awt awaitTour when finished do die
33     behaviour die vanish
```

Task: Q09ATNW (2/2)

Task (continuation)

Task 1: In the following text area, enter the code (one or more lines) that should replace the comment "TASK 1: SOME TEXT TO BE ADDED HERE" in the complete program above.

Enter answer here

Task 2: In the following text area, enter the code (one or more lines) that should replace the comment " TASK 2: SOME TEXT TO BE ADDED HERE " in the complete program above.

Enter answer here

Task 3: In the following text area, enter the code (one or more lines) that should replace the comment " TASK 3: SOME TEXT TO BE ADDED HERE " in the complete program above.

Enter answer here

Correct solution

Task 1:

```
15 n6 at (8, -6) hasDemand stuff units 50 earliestTime 40 latestTime 150 serviceTime 10
```

Task 2:

```
23 road6 from n0 to n6
```

Task 3:

```
29 highway5 from n1 to n6
30 highway6 from n2 to n6
```

Evaluation 2020, 2021

Non-attempt iff three tasks were not answered.

Task 1: 6 Points.
Task 2: 2 Points.
Task 3: 2 Points.

Task: Q09ATAG (1/4)

Introduction

In this task, you'll find an incomplete program together with a visual representation of the network represented by this program. Complete the program in a way so that the target network and agent behaviour visualised at the bottom of the page result. Note that the modelled behaviour, i.e. the tour, must be a likely outcome of the agent behaviour that you modelled in the program.

Note: In this task, there are some nodes for which demands, time windows, and service times are to be defined even though the vehicle must not service them! Navigation nodes (as well as those customer not supposed to be serviced) can be visited by the agent (vehicle) but they will not receive a delivery. Find a way to program this.

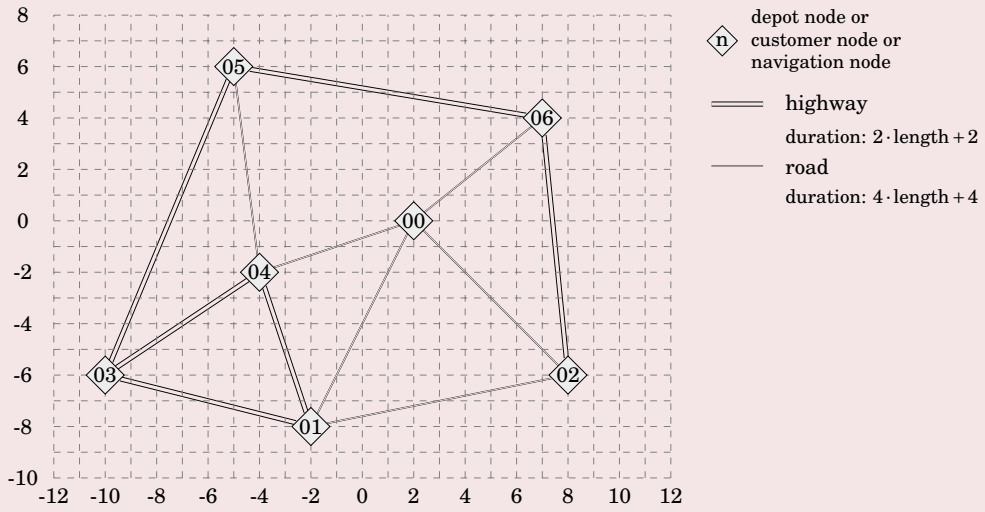
Task

```
1  model q09atag
2  products
3    stuff weight 1.0
4  functions
5    durationFunction roadFunction  4 * length + 4
6    durationFunction highwayFunction  2 * length + 2
7  network
8    nodes
9    // TASK 1 BEGINNING
10   n0 at (2, 0)
11   n1 at (-2, -8) hasDemand stuff units 30 earliestTime 10 latestTime 90 serviceTime 5
12   n2 at (8, -6)
13   n3 at (-10, -6)
14   n4 at (-4, -2)
15   n5 at (-5, 6)
16   n6 at (7, 4)
17   // TASK 1 END
18   edges
19     group roadGroup function roadFunction members
20       road1 from n0 to n6
21       road2 from n0 to n1
22       road3 from n0 to n4
23       road4 from n0 to n2
24       road5 from n1 to n2
25       road6 from n4 to n5
26     group highwayGroup function highwayFunction members
27       highway1 from n3 to n5
28       highway2 from n1 to n3
29       highway3 from n1 to n4
30       highway4 from n4 to n3
31       highway5 from n6 to n5
32       highway6 from n6 to n2
33 agentTypes
34   // TASK 2 BEGINNING
35   agentType vehicles
36   behaviour awt awaitTour when finished do die
37   behaviour die vanish
38   // TASK 2 END
```

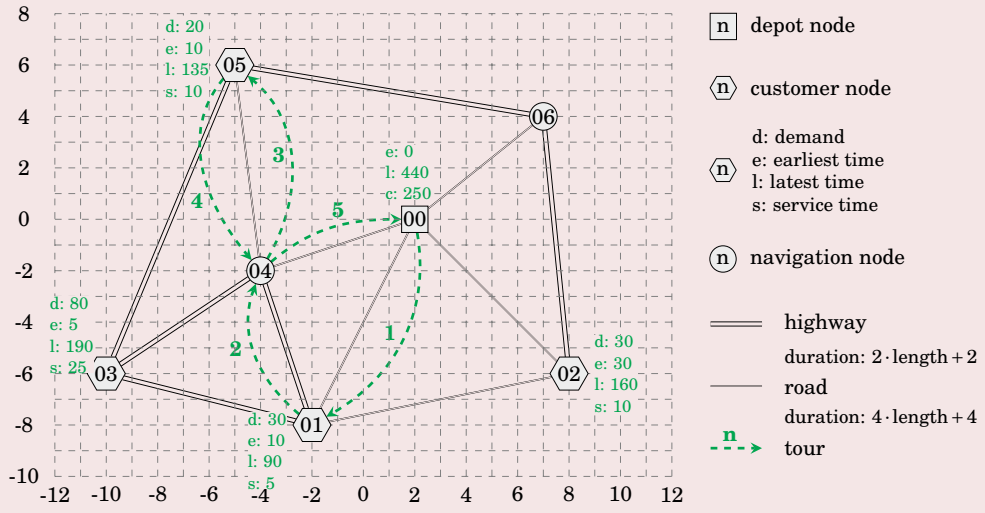
Task: Q09ATAG (2/4)

Task (continuation)

Current state



Target state



Task: Q09ATAG (3/4)

Task (continuation)

Task 1: From the listing above, copy and paste the code betweenthe comment "TASK 1 : BEGINNING" and the comment "TASK 1 END". Complete the copied code for the network (including depot and customer definitions) so that it conforms to the target state depicted in the illustration.

Enter answer here

Task 2: From the listing above, copy and paste the code betweenthe comment "TASK 2 : BEGINNING" and the comment "TASK 2 END". Complete the copied code for the network (including depot and customer definitions) so that it conforms to the target state depicted in the illustration.

Enter answer here

Correct solution

Task 1:

```
10 n0 at (2, 0) isDepot stuff sprouts vehicles customers n1, n5 at 0 latestTime 440
11 n1 at (-2, -8) hasDemand stuff units 30 earliestTime 10 latestTime 90 serviceTime 5
12 n2 at (8, -6) hasDemand stuff units 30 earliestTime 30 latestTime 160 serviceTime 10
13 n3 at (-10, -6) hasDemand stuff units 80 earliestTime 5 latestTime 190 serviceTime 25
14 n4 at (-4, -2)
15 n5 at (-5, 6) hasDemand stuff units 20 earliestTime 10 latestTime 135 serviceTime 10
16 n6 at (7, 4)
```

Task 2:

```
33 agentType vehicles congestionFactor 0 maxWeight 250
34       behaviour awt awaitTour when finished do die
35       behaviour die vanish
```

Task: Q09ATAG (4/4)

Evaluation 2020

Due to a mistake in the creation of the question, the intended answer for the second task, i.e. the addition of '`maxWeight 250`' was already given in the text. For this reason, the second part of this question was removed from the evaluation of the study for both Athos and JSprit.

Task 1: 10 Points.
Task 2: Not evaluated

Evaluation 2022

Non-attempt iff Task1 was not answered.

Task 1: 8 Points. Wrong depot definition and missing customers -4 P. (but +1 Point, if latestTime + value correctly specified, i.e. in total - 3 P.); Wrong depot definition but correc product and correct agent type -3 P. (cannot be combined with latest time +1!); wrong product name -2 P.; completely missing depot definition -4 P.; wrong customer declaration (too many, too few) -2 P.; wrong product and wrong customer declaration -3 P.; wrong code -8 P.; wrong value for demand or time -1 P.; completely wrong demand specification -4 P.; wrong demand specification (one demand specificatoon completely right) -2 P.; syntax error -1 P.; blank space missing (syntax error) -1 P.; missing latest time -2 P.; missing code -1 P.

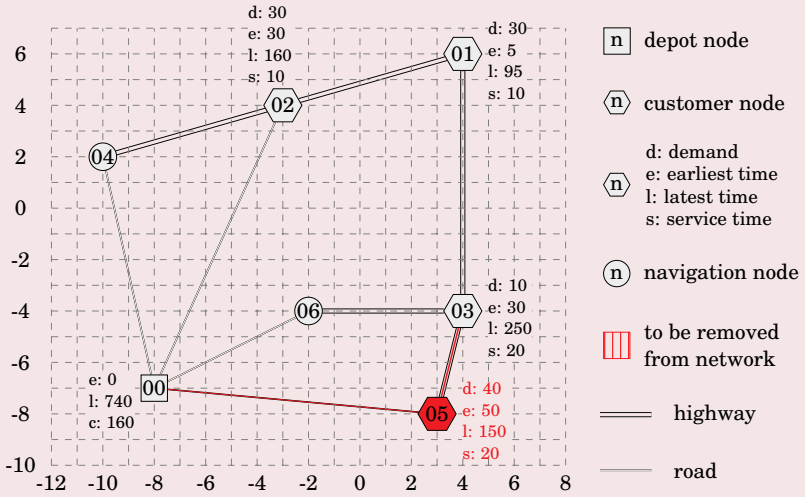
Task 2: 2 Points. Missing maxWeight -2 P.; wrong maximum weight value -1 P.; syntax error -1 P. (-2 P. max.),; completely wrong code -2 P.; missing code, e.g. behaviour (but maxWeight specificaiton correct) 1 P.; changed agent name (no deduction) (-0 P.); wrong keyword (-1 P.); superfluous code -1 P.

Task: Q10ATNW (1/2)

Introduction

In this task, you will find an illustration of a network comprised of highways, roads, navigation and customer nodes. In addition, you will find a program that corresponds to this illustration.In the illustration, some elements (e.g. nodes, demands, highways, etc.) are drawn in red color. These are the elements that are to be removed from the program.
Note: At the bottom of the page is a text area. Copy and paste the code that corresponds to the elements that need to be deleted into this text area. The order in which you paste the elements is not important.

Task




```
model q6nw [xmin -15 xmax 25 ymin -10 ymax 10]
products
  stuff weight 1.0
functions
  durationFunction roadFunction  4 * length + 4
  durationFunction highwayFunction  2 * length + 2
network
  nodes
    n0 at (-8, -7) isDepot stuff sprouts vehicles latestTime 740
    n1 at (4, 6) hasDemand stuff units 30 earliestTime 5 latestTime 95 serviceTime 10
    n2 at (-3, 4) hasDemand stuff units 30 earliestTime 30 latestTime 160 serviceTime 10
    n3 at (4, -4) hasDemand stuff units 10 earliestTime 30 latestTime 250 serviceTime 20
    n4 at (-10, 2)
    n5 at (3, -8) hasDemand stuff units 40 earliestTime 50 latestTime 150 serviceTime 20
    n6 at (-2, -4)
  edges
    group roadGroup function roadFunction members
      road1 from n0 to n4
      road2 from n0 to n2
      road3 from n0 to n6
      road4 from n0 to n5
    group highwayGroup function highwayFunction members
      highway1 from n4 to n2
      highway2 from n2 to n1
      highway3 from n1 to n3
      highway4 from n6 to n3
      highway5 from n3 to n5
agentTypes
  agentType vehicles maxWeight 160
    behaviour awt awaitTour when finished do die
    behaviour die vanish
```

Task: Q10ATNW (2/2)

Task (continuation)

From above Athos model, copy those lines that need to be deleted and paste them in the following text area (in an arbitrary order).

Enter answer here

Correct solution

```
14 n5 at (3, -8) hasDemand stuff units 40 earliestTime 50 latestTime 150 serviceTime 20
21 highway5 from n3 to n5
27 road4 from n0 to n5
```

Evaluation 2020, 2021

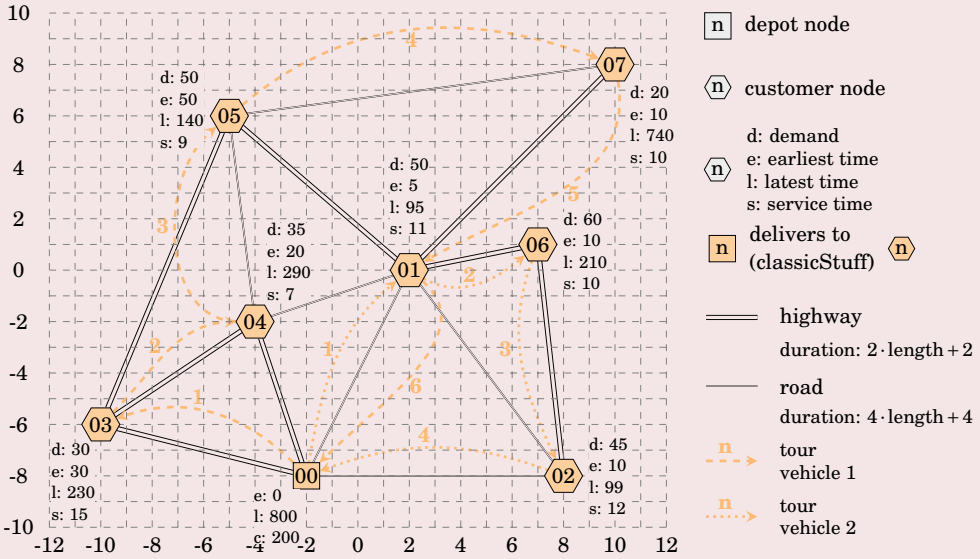
Scheme: 10 Points. Missing removal (of either customer, road or highway): -5 Points.

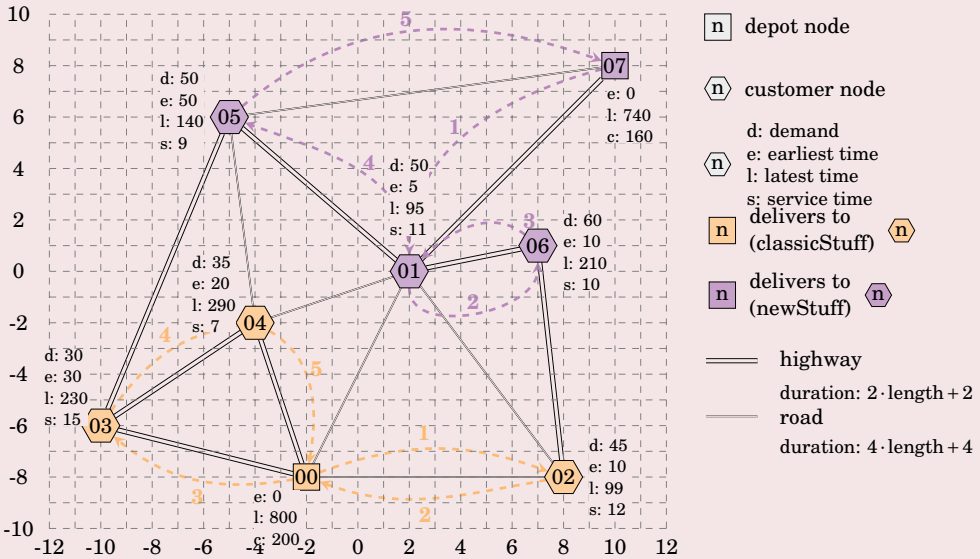
Task: Q11ATALL (1/3)

Introduction

In this task, you will find two graphical representations of networks that are comprised of highways, roads, navigation nodes and customer nodes.
In the first graphical representation you will find one depot together with **seven customer nodes.** The customer nodes are visited by two different tours.
After the two graphical network representations, you will find a program that corresponds **to the first graphical network representation** (i.e. it describes this representation).
The second graphical network representation displays the target state in which you are to transform the program: One of the customer nodes was transformed into a depot from which a new product is delivered to some of the customers. These customers only have a demand for the new product and do no longer require the old one (in other words, they are only supplied by one depot).
The program features comments that mark the beginning and the end of program sections that must be modified in order to transform the program into the target state. At the end of the page, there are corresponding text areas, in which you copy, paste and modify the original code in a suitable way.

Task





Task: Q11ATALL (2/3)

Task (continuation)

```
model q11AllStartState
products
  p0 weight 1.0
  p1 1.0
functions
  durationFunction roadFunction  4 * length + 4
  durationFunction highwayFunction  2 * length + 2
network
  nodes
    // TASK 1 BEGINNING
    n0 at (-2,-8) isDepot p0 sprouts vDepot0 customers n1,n2,n3,n4,n5,n6,n7 at 0 latestTime 800
    n1 at (2, 0) hasDemand p0 units 50 earliestTime 5 latestTime 95 serviceTime 11
    n2 at (8, -8) hasDemand p0 units 45 earliestTime 10 latestTime 99 serviceTime 12
    n3 at (-10, -6) hasDemand p0 units 30 earliestTime 30 latestTime 230 serviceTime 15
    n4 at (-4, -2) hasDemand p0 units 35 earliestTime 20 latestTime 290 serviceTime 7
    n5 at (-5, 6) hasDemand p0 units 50 earliestTime 50 latestTime 140 serviceTime 9
    n6 at (7, 1) hasDemand p0 units 60 earliestTime 10 latestTime 210 serviceTime 10
    n7 at (10, 8) hasDemand p0 units 20  earliestTime 10 latestTime 740 serviceTime 10
    // TASK 1 END
  edges
    group roadGroup function roadFunction members
      road1 from n5 to n7
      road2 from n4 to n5
      road3 from n4 to n1
      road4 from n0 to n1
      road5 from n0 to n2
      road6 from n1 to n2
    group highwayGroup function highwayFunction members
      highway1 from n3 to n5
      highway2 from n3 to n4
      highway3 from n0 to n3
      highway4 from n0 to n4
      highway5 from n1 to n5
      highway6 from n2 to n6
      highway7 from n1 to n6
      highway8 from n1 to n7
agentTypes
  // TASK 2 BEGINNING
  agentType vDepot0 congestionFactor 0 maxWeight 200
    behaviour awt awaitTour when finished do die
    behaviour die vanish
  // TASK 2 END
```

Task 1: From the listing above, copy and paste the code betweenthe comment "TASK 1 : BEGINNING" and the comment "TASK 1 END". Complete / modify the copied code for the network (including depot and customer definitions) so that it conforms to the target state depicted in the illustration.

Enter answer here

Task: Q11ATALL (3/3)

Task (continuation)

Task 2: From the listing above, copy and paste the code betweenthe comment "TASK 2 : BEGINNING" and the comment "TASK 2 END". Complete / modify the copied code for the network (including depot and customer definitions) so that it conforms to the target state depicted in the illustration.

Enter answer here

Correct solution

Task 1:

```
11 n0 at (-2, -8) isDepot p0 sprouts vDepot0 customers n2, n3, n4 at 0 latestTime 800
12 n1 at (2, 0) hasDemand p1 units 50 earliestTime 5 latestTime 95 serviceTime 11
13 n2 at (8, -8) hasDemand p0 units 45 earliestTime 10 latestTime 99 serviceTime 12
14 n3 at (-10, -6) hasDemand p0 units 30 earliestTime 30 latestTime 230 serviceTime 15
15 n4 at (-4, -2) hasDemand p0 units 35 earliestTime 20 latestTime 290 serviceTime 7
16 n5 at (-5, 6) hasDemand p1 units 50 earliestTime 50 latestTime 140 serviceTime 9
17 n6 at (7, 1) hasDemand p1 units 60 earliestTime 10 latestTime 210 serviceTime 10
18 n7 at (10, 8) isDepot p1 sprouts vDepot1 customers n1, n5, n6 at 0 latestTime 740
```

Task 2:

```
42 agentType vDepot1 congestionFactor 0 maxWeight 160
43       behaviour awt awaitTour when finished do die
44       behaviour die vanish
```

Evaluation 2020

Non-attempt iff both tasks were not answered.

Task 1: 6 Points.
Task 2: 4 Points.

Evaluation 2021

Non-attempt iff both tasks were not answered.

The attribution of points was modified. For Athos, the second task required considerably less effort and participants could make less mistakes compared to the first task. With the corresponding JSprit question, both tasks were of similar difficulty.

Task 1: 7 Points.
Task 2: 3 Points.

C.3 JSprit tasks

Task: Q01JSNW (1/2)

Introduction

In this task you are given a program that contains a few syntactical mistakes. Your task is to spot and report these mistakes.

Task

☐ Line 1	☐ Line 2	☐ Line 3	☐ Line 4
☐ Line 5	☐ Line 6	☐ Line 7	☐ Line 8
☐ Line 9	☐ Line 10	☐ Line 11	☐ Line 12
☐ Line 13	☐ Line 14	☐ Line 15	☐ Line 16
☐ Line 17	☐ Line 18	☐ Line 19	☐ Line 20
☐ Line 21			

```
1  public class Q01JSNW {
2
3    private int final static STUFF = 0;
4
5    public static void main(String[] args){
6      Location n0 = create Location(5, 1);
7      Location n1 = Location.newInstance(10, 9);
8      Location n2 = Location.createCustomerAt("12,1");
9
10     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance;
11
12     costMatrixBuilder.addTransportTime(n0,n1, 2 * EuclideanDistanceCalculator
13       .calculateDistance(n0.getCoordinate(),n1.getCoordinate()) + 2);
14     costMatrixBuilder.addTransportTime(n0, n2, 2 * EuclideanDistanceCalculator
15       .calculateDistance(n0.getCoordinate(),n2.getCoordinate() + 2);
16     costMatrixBuilder.addTransportTime(n1, n2, 4 * EuclideanDistanceCalculator
17       .calculateDistance(n1.getCoordinate(),n2.getCoordinate())+ 4);
18     costMatrixBuilder.addTransportTime(n2 n1, 4 * EuclideanDistanceCalculator
19       .calculateDistance(n2.getCoordinate(),n1.getCoordinate())+ 4);
20   }
21 }
```

Task: Q01JSNW (2/2)

Correct solution

☑Line 6, ☑Line 8, ☑Line 10, ☑Line 15, ☑Line 18

Evaluation 2020

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	2
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Evaluation 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q01JSAG

Introduction

In this task you are given a program that contains a few syntactical mistakes. Your task is to spot and report these mistakes.

Task

☐ Line 1	☐ Line 2	☐ Line 3	☐ Line 4
☐ Line 5	☐ Line 6	☐ Line 7	☐ Line 8
☐ Line 9			

```
1 VehicleType type1 = VehicleTypeImpl.Builder
2     .newInstance(type1).addCapacityDimension(0, 130)
3     .build();
4 VehicleType type2 = VehicleTypeImpl.Builder
5     .newInstance("type2").addCapacityDimension(0, 140)
6     .instantiate();
7 VehicleType type3 = VehicleTypeImpl.Builder
8     .newInstance("type3").addCapacityDimension(150)
9     .build();
```

Correct solution

☑ Line 2, ☑ Line 6, ☑ Line 8

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q02JSAG (1/2)

Introduction

In this task you will find a program with a gap. Additionally, you are presented four code snippets that can be used to fill the gap. However, some of these snippets do not make sense (either for themselves or in the context of the completed program). It is your task to find the nonsensical snippets and report them.

Task

Below you see a VRPTW modelled with JSprit. In lines 92 – 95, the definition of a depot is required. Which of the proposed snippets given at the bottom of this page are semantically incorrect (in other words: which of the four snippets do not make complete sense)?

☐ Snippet 1	☐ Snippet 2	☐ Snippet 3	☐ Snippet 4

```
1  public class Q02JSAG {
2    public static int STUFF = 0;
3
4    public static void main(String[] args) {
5      VehicleRoutingProblem.Builder vrpBuilder
6        = VehicleRoutingProblem.Builder.newInstance();
7
8      Location n0 = Location.newInstance(1,1);
9
10     Service n1 = Service.Builder.newInstance("n1")
11       .setLocation(Location.newInstance(1, 8))
12       .addSizeDimension(STUFF, 30)
13       .build();
14
15     Service n2 = Service.Builder.newInstance("n2")
16       .setLocation(Location.newInstance(2, 11))
17       .addSizeDimension(STUFF, 30)
18       .build();
19
20     Location n3 = Location.newInstance(4, 6);
21
22     Service n4 = Service.Builder.newInstance("n4")
23       .setLocation(Location.newInstance(5, 12))
24       .addSizeDimension(STUFF, 30)
25       .build();
26
27     Service n5 = Service.Builder.newInstance("n5")
28       .setLocation(Location.newInstance(8, 11))
29       .addSizeDimension(STUFF, 30)
30       .build();
31
32     Service n6 = Service.Builder.newInstance("n6")
33       .setLocation(Location.newInstance(8, 7))
34       .addSizeDimension(STUFF, 30)
35       .build();
36
37     Location n7 = Location.newInstance(13, 12);
38
39     Location n8 = Location.newInstance(9, 5);
40
41     Location n9 = Location.newInstance(13, 1);
42
```

Task: Q02JSAG (2/2)

Task (continuation)

```
43      IncompleteCostMatrix.Builder costMatrix = IncompleteCostMatrix.Builder.newInstance();
44
45      Set<RelationKey> lcfgroup = new HashSet<>();
46      lcfgroup.add(RelationKey.newKey(n0, n1));
47      lcfgroup.add(RelationKey.newKey(n1, n2 ));
48      lcfgroup.add(RelationKey.newKey(n2, n4));
49      lcfgroup.add(RelationKey.newKey(n4, n5));
50      lcfgroup.add(RelationKey.newKey(n6, n5));
51      lcfgroup.add(RelationKey.newKey(n7, n4));
52      lcfgroup.add(RelationKey.newKey(n7, n9));
53      lcfgroup.add(RelationKey.newKey(n9, n0));
54      lcfgroup.add(RelationKey.newKey(n9, n8));
55      lcfgroup.add(RelationKey.newKey(n8, n6));
56      lcfgroup.add(RelationKey.newKey(n5, n7));
57
58      for(RelationKey key : lcfgroup)
59        costMatrix.addTransportTime(key.from, key.to, roadFunction(key.from, key.to, 2.0));
60
61      Set<RelationKey> hcfgroup = new HashSet<>();
62      hcfgroup.add(RelationKey.newKey(n5, n3));
63      hcfgroup.add(RelationKey.newKey(n3, n0));
64
65      for(RelationKey key : hcfgroup)
66        costMatrix.addTransportTime(key.from, key.to, roadFunction(key.from, key.to, 4.0));
67
68      costMatrix.completeTransportTimeMatrix();
69
70      IncompleteCostMatrix cm = costMatrix.build();
71      vrpBuilder.setRoutingCost(cm);
72
73      VehicleType deliverType = VehicleTypeImpl.Builder.newInstance("vehicles00")
74        .addCapacityDimension(0, 180)
75        .build();
76
77      Vehicle vehicle = VehicleImpl.Builder.newInstance("vehicle")
78        .setType(deliverType).setStartLocation(n0).build();
79      vrpBuilder.addVehicle(vehicle);
80      vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n4, n5, n6));
81
82      VehicleRoutingProblem vrp = vrpBuilder.build();
83    }
84
85    public static double roadFunction(Location end1, Location end2, Double cfactor) {
86      return EuclideanDistanceCalculator
87      .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + cfactor;
88    }
89  }
```

Correct solution

☑Snippet 1, ☑Snippet 2, ☑Snippet 3

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q03JSALL (1/8)

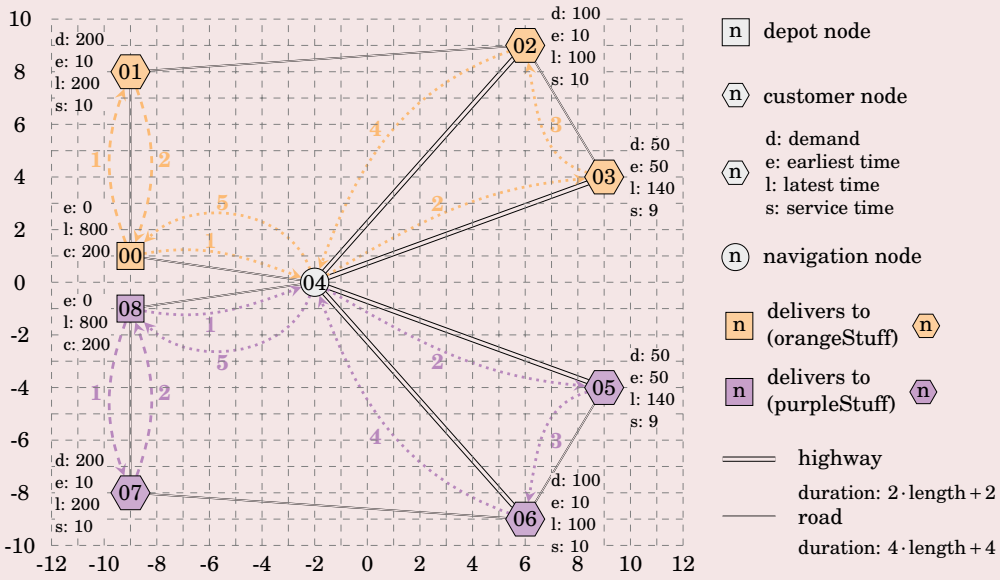
Introduction

In this task you see the illustration of a Network (comprised of customers, demands, roads / highways etc.). In addition, the illustration also shows optimised vehicle routes for a VRP based on the illustrated network. Your task is to determine which of the three models / programs corresponds to the illustrated network.

Task

Which of the three programs corresponds to the illustration?
Wrong answers may feature incorrect customer, depot, road / street, or vehicle definitions.

○Program 1	○Program 2	○Program 3



Task: Q03JSALL (2/8)

Task (continuation)

Program 1

```
1  public class Q03JSALLProgram1 {
2    public static final int ORANGE_STUFF = 0;
3    public static final int PURPLE_STUFF = 1;
4
5    public static void main(String[] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     Location n0 = Location.newInstance(-9, 1);
12
13     Delivery n1 = Delivery.Builder.newInstance("n1")
14       .setLocation(Location.newInstance(-9, 8))
15       .addSizeDimension(ORANGE_STUFF, 200)
16       .setTimeWindow(TimeWindow.newInstance(10, 200))
17       .setServiceTime(10)
18       .build();
19
20     Delivery n2 = Delivery.Builder.newInstance("n2")
21       .setLocation(Location.newInstance(6, 9))
22       .addSizeDimension(ORANGE_STUFF, 100)
23       .setTimeWindow(TimeWindow.newInstance(10, 100))
24       .setServiceTime(10)
25       .build();
26
27     Delivery n3 = Delivery.Builder.newInstance("n3")
28       .setLocation(Location.newInstance(9, 4))
29       .addSizeDimension(ORANGE_STUFF, 50)
30       .setTimeWindow(TimeWindow.newInstance(50, 140))
31       .setServiceTime(9)
32       .build();
33
34     Location n4 = Location.newInstance(-2, 0);
35
36     Delivery n5 = Delivery.Builder.newInstance("n5")
37       .setLocation(Location.newInstance(9, -4))
38       .addSizeDimension(PURPLE_STUFF, 50)
39       .setTimeWindow(TimeWindow.newInstance(50, 140))
40       .setServiceTime(9)
41       .build();
42
43     Delivery n6 = Delivery.Builder.newInstance("n6")
44       .setLocation(Location.newInstance(6, -9))
45       .addSizeDimension(PURPLE_STUFF, 100)
46       .setTimeWindow(TimeWindow.newInstance(10, 100))
47       .setServiceTime(10)
48       .build();
49
50     Delivery n7 = Delivery.Builder.newInstance("n7")
51       .setLocation(Location.newInstance(-9, -8))
52       .addSizeDimension(PURPLE_STUFF, 200)
53       .setTimeWindow(TimeWindow.newInstance(10, 200))
54       .setServiceTime(10)
55       .build();
56
57     Location n8 = Location.newInstance(-9, -1);
58
59     vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n5, n6, n7));
60
```

Task: Q03JSALL (3/8)

Task (continuation)

```
61      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
62      // --- roads
63      Set<RelationKey> roads = new HashSet<>();
64      roads.add(RelationKey.newKey(n2, n4));
65      roads.add(RelationKey.newKey(n3, n4));
66      roads.add(RelationKey.newKey(n6, n4));
67      roads.add(RelationKey.newKey(n5, n4));
68
69      for(RelationKey key : roads)
70        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
71
72      // --- highways
73      Set<RelationKey> highways = new HashSet<>();
74      highways.add(RelationKey.newKey(n0, n1));
75      highways.add(RelationKey.newKey(n0, n4));
76      highways.add(RelationKey.newKey(n1, n2));
77      highways.add(RelationKey.newKey(n2, n3));
78      highways.add(RelationKey.newKey(n8, n7));
79      highways.add(RelationKey.newKey(n8, n4));
80      highways.add(RelationKey.newKey(n7, n6));
81      highways.add(RelationKey.newKey(n5, n6));
82
83      for(RelationKey key : highways)
84        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
85
86      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
87      vrpBuilder.setRoutingCost(cm);
88
89      // Vehicle type definition
90      VehicleType orangeStuffType = VehicleTypeImpl.Builder.newInstance("orangeStuffType")
91        .addCapacityDimension(ORANGE_STUFF, 200).build();
92      VehicleType purpleStuffType = VehicleTypeImpl.Builder.newInstance("purpleStuffType")
93        .addCapacityDimension(PURPLE_STUFF, 200).build();
94
95      // Vehicle instance definition
96      VehicleImpl orangeStuffInstance = VehicleImpl.Builder.newInstance("orangeStuffInstance")
97        .setType(orangeStuffType)
98        .setStartLocation(n0)
99        .build();
100     VehicleImpl purpleStuffInstance = VehicleImpl.Builder.newInstance("purpleStuffInstance")
101       .setType(purpleStuffType)
102       .setStartLocation(n8)
103       .build();
104
105     // Add vehicle instance to the problem
106     vrpBuilder.addVehicle(orangeStuffInstance);
107     vrpBuilder.addVehicle(purpleStuffInstance);
108
109     VehicleRoutingProblem vrp = vrpBuilder.build();
110   }
111
112   public static double highwayFunction(Location end1, Location end2) {
113     return 2 * EuclideanDistanceCalculator
114       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 2;
115   }
116
117   public static double roadFunction(Location end1, Location end2) {
118     return 4 * EuclideanDistanceCalculator
119       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 4;
120   }
121 }
```

Task: Q03JSALL (4/8)

Task (continuation)

Program 2

```
1  public class Q03JSALLProgram2 {
2    public static final int ORANGE_STUFF = 0;
3    public static final int PURPLE_STUFF = 1;
4
5    public static void main(String[] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     Location n0 = Location.newInstance(-9, 1);
12
13     Delivery n1 = Delivery.Builder.newInstance("n1")
14       .setLocation(Location.newInstance(-9, 8))
15       .addSizeDimension(ORANGE_STUFF, 200)
16       .setTimeWindow(TimeWindow.newInstance(10, 200))
17       .setServiceTime(10)
18       .build();
19
20     Delivery n2 = Delivery.Builder.newInstance("n2")
21       .setLocation(Location.newInstance(6, 9))
22       .addSizeDimension(ORANGE_STUFF, 100)
23       .setTimeWindow(TimeWindow.newInstance(10, 100))
24       .setServiceTime(10)
25       .build();
26
27     Delivery n3 = Delivery.Builder.newInstance("n3")
28       .setLocation(Location.newInstance(9, 4))
29       .addSizeDimension(ORANGE_STUFF, 50)
30       .setTimeWindow(TimeWindow.newInstance(50, 140))
31       .setServiceTime(9)
32       .build();
33
34     Location n4 = Location.newInstance(-2, 0);
35
36     Delivery n5 = Delivery.Builder.newInstance("n5")
37       .setLocation(Location.newInstance(9, -4))
38       .addSizeDimension(PURPLE_STUFF, 50)
39       .setTimeWindow(TimeWindow.newInstance(50, 140))
40       .setServiceTime(9)
41       .build();
42
43     Delivery n6 = Delivery.Builder.newInstance("n6")
44       .setLocation(Location.newInstance(6, -9))
45       .addSizeDimension(PURPLE_STUFF, 100)
46       .setTimeWindow(TimeWindow.newInstance(10, 100))
47       .setServiceTime(10)
48       .build();
49
50     Delivery n7 = Delivery.Builder.newInstance("n7")
51       .setLocation(Location.newInstance(-9, -8))
52       .addSizeDimension(PURPLE_STUFF, 200)
53       .setTimeWindow(TimeWindow.newInstance(10, 200))
54       .setServiceTime(10)
55       .build();
56
57     Location n8 = Location.newInstance(-9, -1);
58
59     vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n5, n6, n7));
60
```

Task: Q03JSALL (5/8)

Task (continuation)

```
61      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
62      // --- roads
63      Set<RelationKey> roads = new HashSet<>();
64      roads.add(RelationKey.newKey(n0, n1));
65      roads.add(RelationKey.newKey(n0, n4));
66      roads.add(RelationKey.newKey(n1, n2));
67      roads.add(RelationKey.newKey(n2, n3));
68      roads.add(RelationKey.newKey(n8, n7));
69      roads.add(RelationKey.newKey(n8, n4));
70      roads.add(RelationKey.newKey(n7, n6));
71      roads.add(RelationKey.newKey(n5, n6));
72
73      for(RelationKey key : roads)
74        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
75
76      // --- highways
77      Set<RelationKey> highways = new HashSet<>();
78      highways.add(RelationKey.newKey(n2, n4));
79      highways.add(RelationKey.newKey(n3, n4));
80      highways.add(RelationKey.newKey(n6, n4));
81      highways.add(RelationKey.newKey(n5, n4));
82
83      for(RelationKey key : highways)
84        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
85
86      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
87      vrpBuilder.setRoutingCost(cm);
88
89      // Vehicle type definition
90      VehicleType orangeStuffType = VehicleTypeImpl.Builder.newInstance("orangeStuffType")
91      .addCapacityDimension(ORANGE_STUFF, 200).build();
92      VehicleType purpleStuffType = VehicleTypeImpl.Builder.newInstance("purpleStuffType")
93      .addCapacityDimension(PURPLE_STUFF, 200).build();
94
95      // Vehicle instance defintion
96      VehicleImpl orangeStuffInstance = VehicleImpl.Builder.newInstance("orangeStuffInstance")
97      .setType(orangeStuffType)
98      .setStartLocation(n8)
99      .build();
100     VehicleImpl purpleStuffInstance = VehicleImpl.Builder.newInstance("purpleStuffInstance")
101     .setType(purpleStuffType)
102     .setStartLocation(n0)
103     .build();
104
105     // Add vehicle instance to the problem
106     vrpBuilder.addVehicle(orangeStuffInstance);
107     vrpBuilder.addVehicle(purpleStuffInstance);
108
109     VehicleRoutingProblem vrp = vrpBuilder.build();
110   }
111
112   public static double highwayFunction(Location end1, Location end2) {
113     return 2 * EuclideanDistanceCalculator
114       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 2;
115   }
116
117   public static double roadFunction(Location end1, Location end2) {
118     return 4 * EuclideanDistanceCalculator
119       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 4;
120   }
121 }
```

Task: Q03JSALL (6/8)

Task (continuation)

Program 3

```
1  public class Q03JSALLProgram3 {
2    public static final int ORANGE_STUFF = 0;
3    public static final int PURPLE_STUFF = 1;
4
5    public static void main(String[] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     Location n0 = Location.newInstance(-9, 1);
12
13     Delivery n1 = Delivery.Builder.newInstance("n1")
14       .setLocation(Location.newInstance(-9, 8))
15       .addSizeDimension(ORANGE_STUFF, 200)
16       .setTimeWindow(TimeWindow.newInstance(10, 200))
17       .setServiceTime(10)
18       .build();
19
20     Delivery n2 = Delivery.Builder.newInstance("n2")
21       .setLocation(Location.newInstance(6, 9))
22       .addSizeDimension(ORANGE_STUFF, 100)
23       .setTimeWindow(TimeWindow.newInstance(10, 100))
24       .setServiceTime(10)
25       .build();
26
27     Delivery n3 = Delivery.Builder.newInstance("n3")
28       .setLocation(Location.newInstance(9, 4))
29       .addSizeDimension(ORANGE_STUFF, 50)
30       .setTimeWindow(TimeWindow.newInstance(50, 140))
31       .setServiceTime(9)
32       .build();
33
34     Location n4 = Location.newInstance(-2, 0);
35
36     Delivery n5 = Delivery.Builder.newInstance("n5")
37       .setLocation(Location.newInstance(9, -4))
38       .addSizeDimension(PURPLE_STUFF, 50)
39       .setTimeWindow(TimeWindow.newInstance(50, 140))
40       .setServiceTime(9)
41       .build();
42
43     Delivery n6 = Delivery.Builder.newInstance("n6")
44       .setLocation(Location.newInstance(6, -9))
45       .addSizeDimension(PURPLE_STUFF, 100)
46       .setTimeWindow(TimeWindow.newInstance(10, 100))
47       .setServiceTime(10)
48       .build();
49
50     Delivery n7 = Delivery.Builder.newInstance("n7")
51       .setLocation(Location.newInstance(-9, -8))
52       .addSizeDimension(PURPLE_STUFF, 200)
53       .setTimeWindow(TimeWindow.newInstance(10, 200))
54       .setServiceTime(10)
55       .build();
56
57     Location n8 = Location.newInstance(-9, -1);
58
59     vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n5, n6, n7));
60
```

Task: Q03JSALL (7/8)

Task (continuation)

```
61      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
62      // --- roads
63      Set<RelationKey> roads = new HashSet<>();
64      roads.add(RelationKey.newKey(n0, n1));
65      roads.add(RelationKey.newKey(n0, n4));
66      roads.add(RelationKey.newKey(n1, n2));
67      roads.add(RelationKey.newKey(n2, n3));
68      roads.add(RelationKey.newKey(n8, n7));
69      roads.add(RelationKey.newKey(n8, n4));
70      roads.add(RelationKey.newKey(n7, n6));
71      roads.add(RelationKey.newKey(n5, n6));
72
73      for(RelationKey key : roads)
74        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
75
76      // --- highways
77      Set<RelationKey> highways = new HashSet<>();
78      highways.add(RelationKey.newKey(n2, n4));
79      highways.add(RelationKey.newKey(n3, n4));
80      highways.add(RelationKey.newKey(n6, n4));
81      highways.add(RelationKey.newKey(n5, n4));
82
83      for(RelationKey key : highways)
84        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
85
86      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
87      vrpBuilder.setRoutingCost(cm);
88
89      // Vehicle type definition
90      VehicleType orangeStuffType = VehicleTypeImpl.Builder.newInstance("orangeStuffType")
91        .addCapacityDimension(ORANGE_STUFF, 200).build();
92      VehicleType purpleStuffType = VehicleTypeImpl.Builder.newInstance("purpleStuffType")
93        .addCapacityDimension(PURPLE_STUFF, 200).build();
94
95      // Vehicle instance defintion
96      VehicleImpl orangeStuffInstance = VehicleImpl.Builder.newInstance("orangeStuffInstance")
97        .setType(orangeStuffType)
98        .setStartLocation(n0)
99        .build();
100     VehicleImpl purpleStuffInstance = VehicleImpl.Builder.newInstance("purpleStuffInstance")
101       .setType(purpleStuffType)
102       .setStartLocation(n8)
103       .build();
104
105     // Add vehicle instance to the problem
106     vrpBuilder.addVehicle(orangeStuffInstance);
107     vrpBuilder.addVehicle(purpleStuffInstance);
108
109     VehicleRoutingProblem vrp = vrpBuilder.build();
110   }
111
112   public static double highwayFunction(Location end1, Location end2) {
113     return 2 * EuclideanDistanceCalculator
114       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 2;
115   }
116
117   public static double roadFunction(Location end1, Location end2) {
118     return 4 * EuclideanDistanceCalculator
119       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 4;
120   }
121 }
```

Task: Q03JSALL (8/8)

Correct solution

◉ Program 3

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q04JSNW (1/4)

Introduction

In this task you are shown a program and four different graphical networks. One of these networks is exactly described (modelled) by the program. For the other three networks the program is not completely right. It is your task to find and report the exactly modelled network.

Task

Which of the above networks results from the given JSprit model?

○Network A	○Network B	○Network C	○Network D

```
1  public class Q04JSNW {
2    public static int STUFF = 0;
3
4    public static void main(String[] args) {
5      VehicleRoutingProblem.Builder vrpBuilder
6        = VehicleRoutingProblem.Builder.newInstance();
7
8      Location n0 = Location.newInstance(0, -6);
9
10     Service n1 = Service.Builder.newInstance("n1")
11       .setLocation(Location.newInstance(-9,4))
12       .addSizeDimension(STUFF, 15)
13       .setTimeWindow(TimeWindow.newInstance(15, 120))
14       .setServiceTime(5)
15       .build();
16
17     Service n2 = Service.Builder.newInstance("n2")
18       .setLocation(Location.newInstance(7,-9))
19       .addSizeDimension(STUFF, 20)
20       .setTimeWindow(TimeWindow.newInstance(10,130))
21       .setServiceTime(7)
22       .build();
23
24     Service n3 = Service.Builder.newInstance("n3")
25       .setLocation(Location.newInstance(8,5))
26       .addSizeDimension(STUFF, 50)
27       .setTimeWindow(TimeWindow.newInstance(20,90))
28       .setServiceTime(10)
29       .build();
30
31     Location n4 = Location.newInstance(2,0);
32
33     Service n5 = Service.Builder.newInstance("n5")
34       .setLocation(Location.newInstance(-2, -1))
35       .addSizeDimension(STUFF, 10)
36       .setTimeWindow(TimeWindow.newInstance(80, 150))
37       .setServiceTime(20)
38       .build();
39
40     Service n6 = Service.Builder.newInstance("n6")
41       .setLocation(Location.newInstance(-8,-6))
42       .addSizeDimension(STUFF,25)
43       .setTimeWindow(TimeWindow.newInstance(90,270))
44       .setServiceTime(5)
45       .build();
46
```

Task: Q04JSNW (2/4)

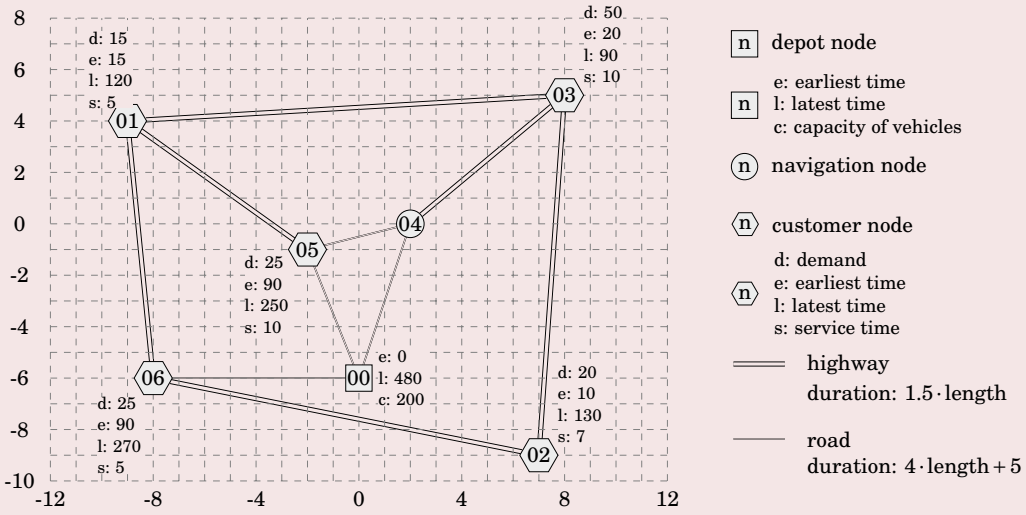
Task (continuation)

```
47      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
48
49      costMatrixBuilder.addTransportTime(n0, n5.getLocation(),
50        roadFunction(n0, n5.getLocation()));
51
52      costMatrixBuilder.addTransportTime(n0,n4,roadFunction(n0,n4));
53
54      costMatrixBuilder.addTransportTime(n5.getLocation(), n4,
55        roadFunction(n5.getLocation(),n4));
56
57      costMatrixBuilder.addTransportTime(n0, n6.getLocation(),
58        roadFunction(n0,n6.getLocation()));
59
60      costMatrixBuilder.addTransportTime(n1.getLocation(), n3.getLocation(),
61        highwayFunction(n1.getLocation(),n3.getLocation()));
62
63      costMatrixBuilder.addTransportTime(n3.getLocation(), n2.getLocation(),
64        highwayFunction(n3.getLocation(), n2.getLocation()));
65
66      costMatrixBuilder.addTransportTime(n2.getLocation(), n6.getLocation(),
67        highwayFunction(n2.getLocation(), n6.getLocation()));
68
69      costMatrixBuilder.addTransportTime(n6.getLocation(), n1.getLocation(),
70        highwayFunction(n6.getLocation(), n1.getLocation()));
71
72      costMatrixBuilder.addTransportTime(n4, n3.getLocation(),
73        highwayFunction(n4,n3.getLocation()));
74
75      costMatrixBuilder.addTransportTime(n5.getLocation(), n1.getLocation(),
76        highwayFunction(n5.getLocation(), n1.getLocation()));
77
78      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
79      vrpBuilder.setRoutingCost(cm);
80
81      VehicleType deliverType = VehicleTypeImpl.Builder.newInstance("vehicles00")
82        .addCapacityDimension(0, 180)
83        .build();
84
85      Vehicle vehicle = VehicleImpl.Builder.newInstance("vehicle")
86        .setType(deliverType).setStartLocation(n0).setLatestArrival(500).build();
87      vrpBuilder.addVehicle(vehicle);
88      vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n5, n6));
89
90      VehicleRoutingProblem vrp = vrpBuilder.build();
91    }
92
93    public static double highwayFunction(Location end1, Location end2) {
94      return 1.5 * EuclideanDistanceCalculator
95        .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
96    }
97
98    public static double roadFunction(Location end1, Location end2) {
99      return 3 * EuclideanDistanceCalculator
100       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) *  + 3;
101   }
102 }
```

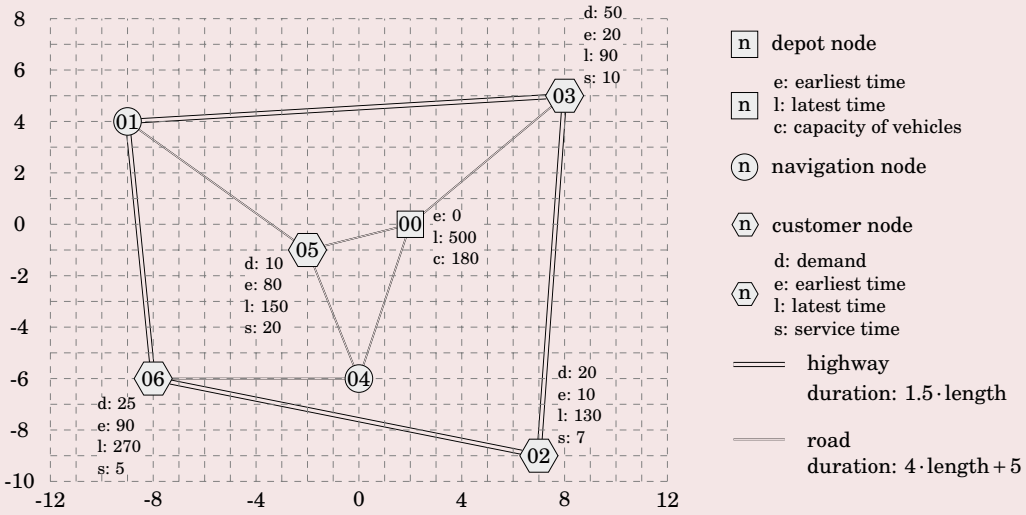
Task: Q04JSNW (3/4)

Task (continuation)

Network A



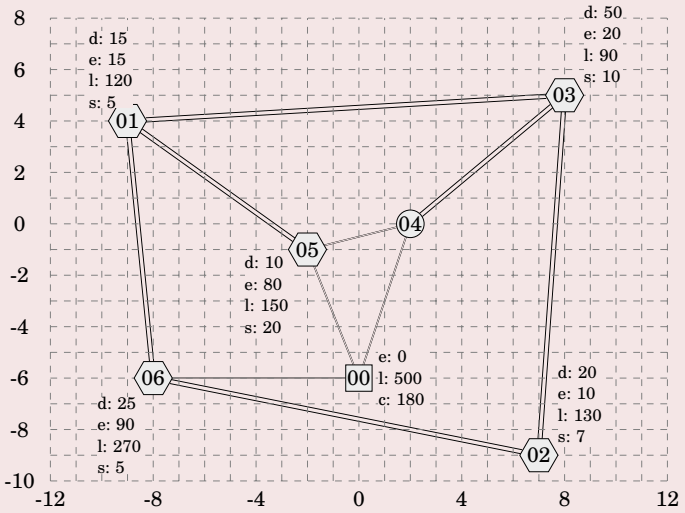
Network B



Task: Q04JSNW (4/4)

Task (continuation)

Network C



n depot node

n e: earliest time
l: latest time
c: capacity of vehicles

(n) navigation node

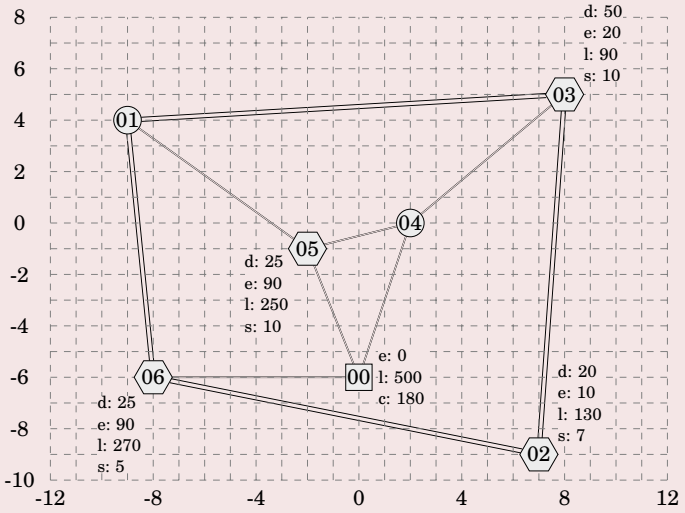
⟨n⟩ customer node

⟨n⟩ d: demand
e: earliest time
l: latest time
s: service time

highway
duration: $1.5 \cdot \text{length}$

road
duration: $3 \cdot \text{length} + 3$

Network D



n depot node

n e: earliest time
l: latest time
c: capacity of vehicles

(n) navigation node

⟨n⟩ customer node

⟨n⟩ d: demand
e: earliest time
l: latest time
s: service time

highway
duration: $1.5 \cdot \text{length}$

road
duration: $3 \cdot \text{length} + 3$

Correct solution

◉ Network C

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q04JSAG (1/4)

Introduction

In this task you are shown a program and four graphical networks on which four different tours are depicted. One of these networks shows a tour that is exactly described (modelled) by the program. For the other three tours the program is not completely right. It is your task to find and report the exactly modelled tour.

NOTE: If a tour step connects two nodes that do not share an edge, this means that the actual path from the start node to the target node of the respective step is not important. However, only nodes of the drawn tour are serviced!

Task

Which of the presented tours for a vehicle may result from the program given below?

○Tour 1	○Tour 2	○Tour 3	○Tour 4

```
1  public class Q04JSAG {
2    public static int STUFF = 0;
3
4    public static void main(String[] args) {
5      VehicleRoutingProblem.Builder vrpBuilder
6        = VehicleRoutingProblem.Builder.newInstance();
7
8      Location n0 = Location.newInstance(-10, -8);
9      Location n1 = Location.newInstance(-8,-4);
10     Service n2 = Service.Builder.newInstance("n2")
11       .setLocation(Location.newInstance(-11,2))
12       .addSizeDimension(STUFF, 20)
13       .setTimeWindow(TimeWindow.newInstance(10, 130))
14       .setServiceTime(7)
15       .build();
16     Location n3 = Location.newInstance(-5,4);
17     Service n4 = Service.Builder.newInstance("n4")
18       .setLocation(Location.newInstance(-8,7))
19       .addSizeDimension(STUFF, 50)
20       .setTimeWindow(TimeWindow.newInstance(20,90))
21       .setServiceTime(10)
22       .build();
23     Service n5 = Service.Builder.newInstance("n5")
24       .setLocation(Location.newInstance(2,0))
25       .addSizeDimension(STUFF, 25)
26       .setTimeWindow(TimeWindow.newInstance(90,250))
27       .setServiceTime(10)
28       .build();
29     Service n6 = Service.Builder.newInstance("n6")
30       .setLocation(Location.newInstance(2,7))
31       .addSizeDimension(STUFF, 25)
32       .setTimeWindow(TimeWindow.newInstance(90,250))
33       .setServiceTime(10)
34       .build();
35     Location n7 = Location.newInstance(6, 5);
36     Location n8 = Location.newInstance(11, 3);
37     Location n9 = Location.newInstance(2,-8);
38     Location n10 = Location.newInstance(8,-6);
39     Location n11 = Location.newInstance(6,-9);
40
```

Task: Q04JSAG (2/4)

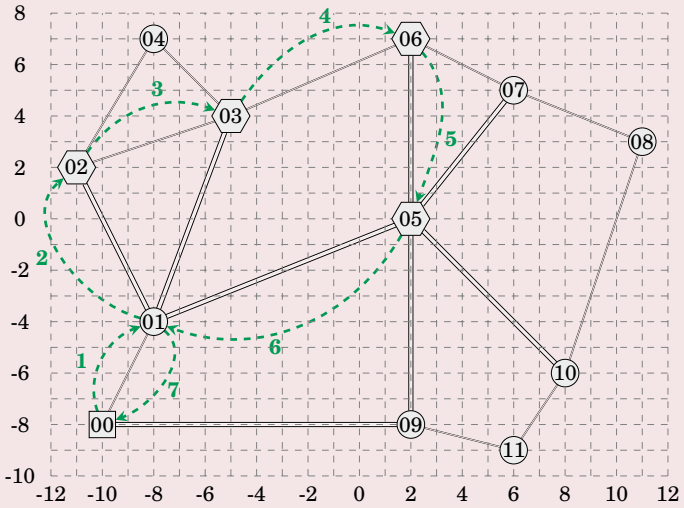
Task (continuation)

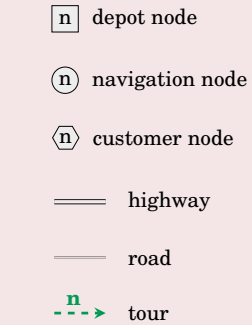
```
41      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
42      costMatrixBuilder.addTransportTime(n0, n1, roadFunction(n0, n1));
43      costMatrixBuilder.addTransportTime(n2.getLocation(), n4.getLocation(),
44        roadFunction(n2.getLocation(), n4.getLocation()));
45      costMatrixBuilder.addTransportTime(n2.getLocation(), n3,
46      roadFunction(n2.getLocation(), n3));
47      costMatrixBuilder.addTransportTime(n4.getLocation(), n3,
48        roadFunction(n4.getLocation(), n3));
49      costMatrixBuilder.addTransportTime(n3, n6.getLocation(),
50        roadFunction(n3, n6.getLocation()));
51      costMatrixBuilder.addTransportTime(n6.getLocation(), n7,
52        roadFunction(n6.getLocation(), n7));
53      costMatrixBuilder.addTransportTime(n7, n8, roadFunction(n7, n8));
54      costMatrixBuilder.addTransportTime(n8, n10, roadFunction(n8, n10));
55      costMatrixBuilder.addTransportTime(n10, n11, roadFunction(n10, n11));
56      costMatrixBuilder.addTransportTime(n11, n9, roadFunction(n11, n9));
57
58      costMatrixBuilder.addTransportTime(n1, n2.getLocation(),
59        highwayFunction(n1, n2.getLocation()));
60      costMatrixBuilder.addTransportTime(n1, n3, highwayFunction(n1, n3));
61      costMatrixBuilder.addTransportTime(n1, n5.getLocation(),
62        highwayFunction(n1, n5.getLocation()));
63      costMatrixBuilder.addTransportTime(n5.getLocation(), n6.getLocation(),
64        highwayFunction(n5.getLocation(), n6.getLocation()));
65      costMatrixBuilder.addTransportTime(n5.getLocation(), n7,
66        highwayFunction(n5.getLocation(), n7));
67      costMatrixBuilder.addTransportTime(n5.getLocation(), n10,
68        highwayFunction(n5.getLocation(), n10));
69      costMatrixBuilder.addTransportTime(n5.getLocation(), n9,
70        highwayFunction(n5.getLocation(), n9));
71      costMatrixBuilder.addTransportTime(n9, n0, highwayFunction(n9, n0));
72
73      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
74      vrpBuilder.setRoutingCost(cm);
75
76      VehicleType deliverType = VehicleTypeImpl.Builder.newInstance("vehicles00")
77        .addCapacityDimension(0, 180)
78        .build();
79
80      Vehicle vehicle = VehicleImpl.Builder.newInstance("vehicle")
81        .setType(deliverType).setStartLocation(n0).setLatestArrival(500).build();
82      vrpBuilder.addVehicle(vehicle);
83      vrpBuilder.addAllJobs(Arrays.asList(n2, n4, n5, n6));
84
85      VehicleRoutingProblem vrp = vrpBuilder.build();
86    }
87
88    public static double highwayFunction(Location end1, Location end2) {
89      return EuclideanDistanceCalculator
90        .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
91    }
92
93    public static double roadFunction(Location end1, Location end2) {
94      return 3 * EuclideanDistanceCalculator
95        .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
96    }
97  }
```

Task: Q04JSAG (3/4)

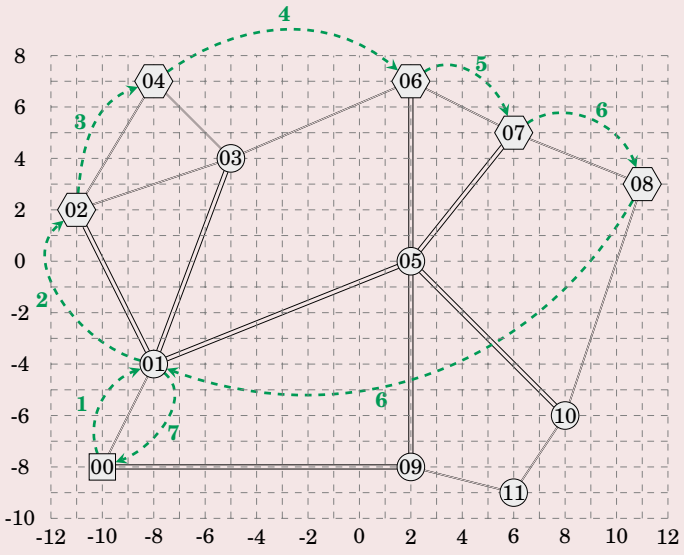
Task (continuation)

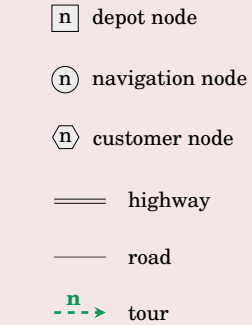
Tour 1





Tour 2

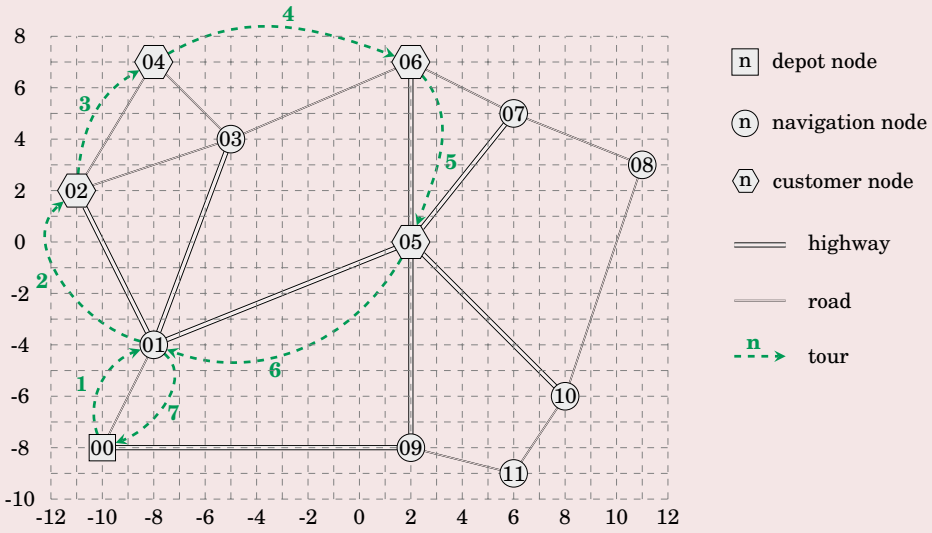




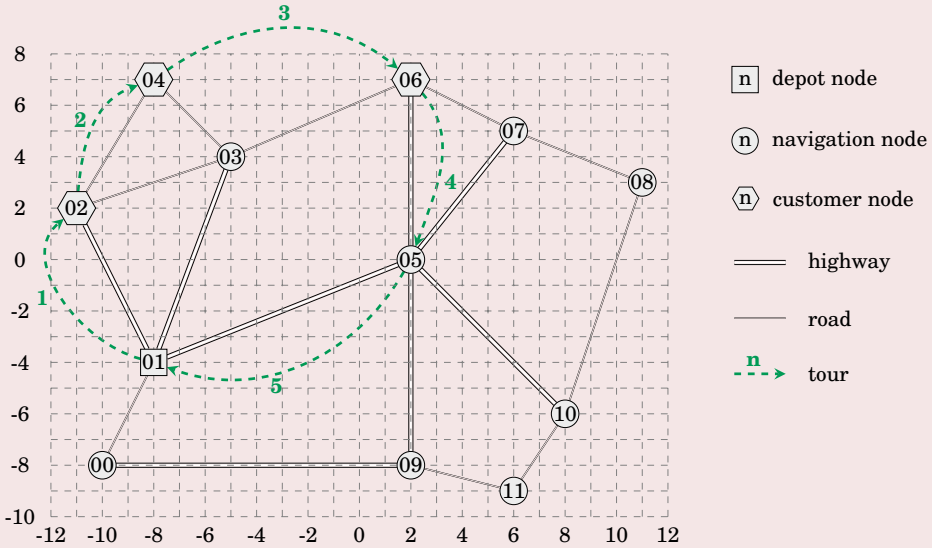
Task: Q04JSAG (4/4)

Task (continuation)

Tour 3



Tour 4



Correct solution

◉ Tour 3

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q05JSNW (1/3)

Introduction

In this task you will find a program and two networks. The first of the two networks is described by the program. If some of the lines of the program are changed, the program is a description of the second network. It is your task to find these lines and report them.

Task

The JSprit model below represents network 1 . What lines of the JSprit model would have to be changed if you wanted to model network 2 (multiple answers possible)?

☐ Line 01	☐ Line 02	☐ Line 03	☐ Line 04	☐ Line 05	☐ Line 06	☐ Line 07
☐ Line 08	☐ Line 09	☐ Line 10	☐ Line 11	☐ Line 12	☐ Line 13	☐ Line 14
☐ Line 15	☐ Line 16	☐ Line 17	☐ Line 18	☐ Line 19	☐ Line 20	☐ Line 21
☐ Line 22	☐ Line 23	☐ Line 24	☐ Line 25	☐ Line 26	☐ Line 27	☐ Line 28
☐ Line 29	☐ Line 30	☐ Line 31	☐ Line 32	☐ Line 33	☐ Line 34	☐ Line 35
☐ Line 36	☐ Line 37	☐ Line 38	☐ Line 39	☐ Line 40	☐ Line 41	☐ Line 42
☐ Line 43	☐ Line 44	☐ Line 45	☐ Line 46	☐ Line 47	☐ Line 48	☐ Line 49
☐ Line 50	☐ Line 51	☐ Line 52	☐ Line 53	☐ Line 54	☐ Line 55	☐ Line 56
☐ Line 57	☐ Line 58	☐ Line 59	☐ Line 60	☐ Line 61	☐ Line 62	☐ Line 63
☐ Line 64	☐ Line 65	☐ Line 66	☐ Line 67	☐ Line 68	☐ Line 69	☐ Line 70
☐ Line 71	☐ Line 72	☐ Line 73	☐ Line 74	☐ Line 75	☐ Line 76	☐ Line 77
☐ Line 78	☐ Line 79	☐ Line 80	☐ Line 81	☐ Line 82	☐ Line 83	☐ Line 84
☐ Line 85	☐ Line 86	☐ Line 87	☐ Line 88	☐ Line 89	☐ Line 90	☐ Line 91
☐ Line 92	☐ Line 93					

```
1  public class Q05JSNW {
2    // Products
3    public static int STUFF = 0;
4
5    public static void main(String[] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     Location n0 = Location.newInstance(-2, -2);
12
13     Location n1 = Location.newInstance(-8, -4);
14
15     Service n2 = Service.Builder.newInstance("n2")
16       .setLocation(Location.newInstance(-8,-1))
17       .addSizeDimension(STUFF, 20)
18       .setTimeWindow(TimeWindow.newInstance(10, 130))
19       .setServiceTime(7)
20       .build();
21
22     Service n3 = Service.Builder.newInstance("n3")
23       .setLocation(Location.newInstance(-5, 4))
24       .addSizeDimension(STUFF, 50)
25       .setTimeWindow(TimeWindow.newInstance(20, 90))
26       .setServiceTime(10)
27       .build();
28
29     Service n4 = Service.Builder.newInstance("n4")
30       .setLocation(Location.newInstance(-2, -6))
31       .addSizeDimension(STUFF, 25)
32       .setTimeWindow(TimeWindow.newInstance(90, 250))
33       .setServiceTime(10)
34       .build();
35
36     Location n5 = Location.newInstance(2, 0);
```

Task: Q05JSNW (2/3)

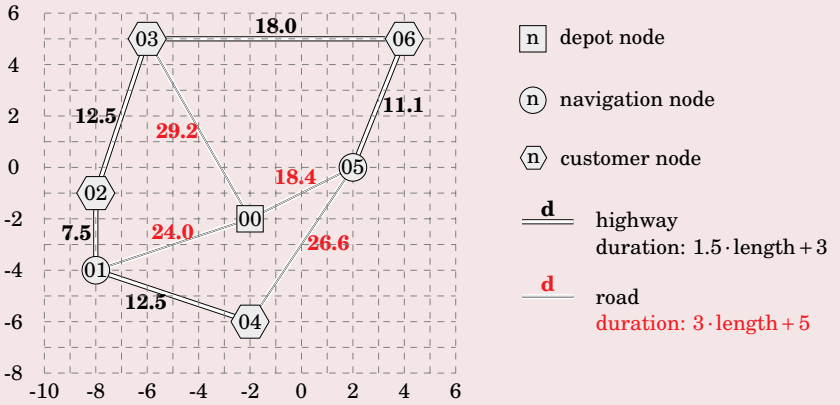
Task (continuation)

```
37
38      Service n6 = Service.Builder.newInstance("n4")
39        .setLocation(Location.newInstance(4, 5))
40        .addSizeDimension(STUFF, 25)
41        .setTimeWindow(TimeWindow.newInstance(90, 250))
42        .setServiceTime(10)
43        .build();
44
45      // --- Edges definition
46      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
47
48      // --- --- roads
49      costMatrixBuilder.addTransportTime(n0, n1, roadFunction(n0, n1));
50
51      costMatrixBuilder.addTransportTime(n0, n3.getLocation(),
52      roadFunction(n0,n3.getLocation()));
53
54      costMatrixBuilder.addTransportTime(n0,n5, roadFunction(n0,n5));
55
56      costMatrixBuilder.addTransportTime(n4.getLocation(), n5,
57      roadFunction(n4.getLocation(), n5));
58
59      // --- --- highways
60      costMatrixBuilder.addTransportTime(n1,n2.getLocation(),
61      highwayFunction(n1, n2.getLocation()));
62
63      costMatrixBuilder.addTransportTime(n2.getLocation(),n3.getLocation(),
64      highwayFunction(n2.getLocation(), n3.getLocation()));
65
66      costMatrixBuilder.addTransportTime(n3.getLocation(),n6.getLocation(),
67      highwayFunction(n3.getLocation(), n6.getLocation()));
68
69      costMatrixBuilder.addTransportTime(n5,n6.getLocation(),
70      highwayFunction(n5, n3.getLocation()));
71
72      costMatrixBuilder.addTransportTime(n1, n4.getLocation(),
73      highwayFunction(n1, n4.getLocation()));
74
75      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
76      vrpBuilder.setRoutingCost(cm);
77
78      vrpBuilder.addAllJobs(Arrays.asList(n2, n3, n4, n6));
79
80      // Vehicle type and vehicle and problem build omitted
81    }
82
83    public static double highwayFunction(Location end1, Location end2) {
84      return 1.5 * EuclideanDistanceCalculator
85        .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
86    }
87
88    public static double roadFunction(Location end1, Location end2) {
89      return 3 * EuclideanDistanceCalculator
90        .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
91    }
92
93  }
```

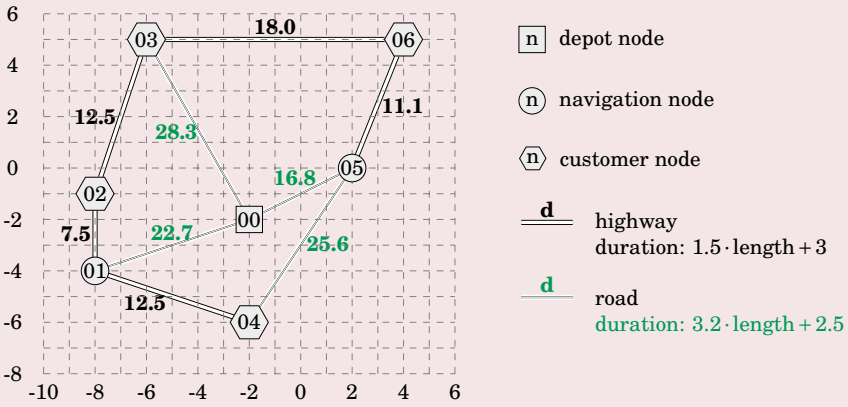
Task: Q05JSNW (3/3)

Task (continuation)

Network 1 (currently modelled)



Network 2 (target state)



Correct solution

☑Line 89, ☑Line 90

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q05JSAG (1/3)

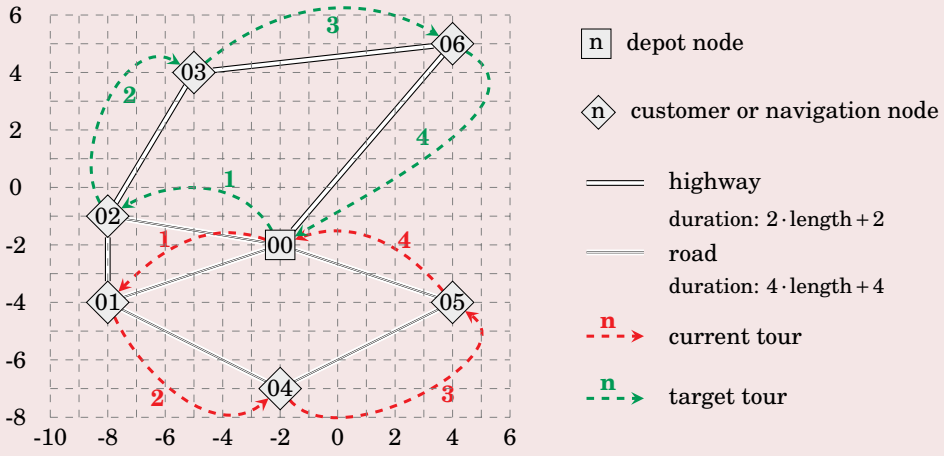
Introduction

In this task, you will find the graphical representation of a network together with a program. The network shows two tours. The program describes the network and a problem for which one of the tours is likely to be optimal. By changing some lines of the program, the program describes a problem for which the other tour is likely to be optimal. It is your task to find and report these lines.

Task

In the picture below, the red tour is a likely result from the problem modelled in the JSprit code given at the bottom. What lines of the JSprit model have to be changed so that the green tour r is likely to result from the model?

☐ Line 01	☐ Line 02	☐ Line 03	☐ Line 04	☐ Line 05	☐ Line 06	☐ Line 07
☐ Line 08	☐ Line 09	☐ Line 10	☐ Line 11	☐ Line 12	☐ Line 13	☐ Line 14
☐ Line 15	☐ Line 16	☐ Line 17	☐ Line 18	☐ Line 19	☐ Line 20	☐ Line 21
☐ Line 22	☐ Line 23	☐ Line 24	☐ Line 25	☐ Line 26	☐ Line 27	☐ Line 28
☐ Line 29	☐ Line 30	☐ Line 31	☐ Line 32	☐ Line 33	☐ Line 34	☐ Line 35
☐ Line 36	☐ Line 37	☐ Line 38	☐ Line 39	☐ Line 40	☐ Line 41	☐ Line 42
☐ Line 43	☐ Line 44	☐ Line 45	☐ Line 46	☐ Line 47	☐ Line 48	☐ Line 49
☐ Line 50	☐ Line 51	☐ Line 52	☐ Line 53	☐ Line 54	☐ Line 55	☐ Line 56
☐ Line 57	☐ Line 58	☐ Line 59	☐ Line 60	☐ Line 61	☐ Line 62	☐ Line 63
☐ Line 64	☐ Line 65	☐ Line 66	☐ Line 67	☐ Line 68	☐ Line 69	☐ Line 70
☐ Line 71	☐ Line 72	☐ Line 73	☐ Line 74	☐ Line 75	☐ Line 76	☐ Line 77
☐ Line 78	☐ Line 79	☐ Line 80				



Task: Q05JSAG (2/3)

Task (continuation)

```
1  public class Q05JSAG {
2    public static int STUFF = 0;
3
4    public static void main(String[] args) {
5      VehicleRoutingProblem.Builder vrpBuilder
6        = VehicleRoutingProblem.Builder.newInstance();
7      Location n0 = Location.newInstance(-2, -2);
8      Service n1 = Service.Builder.newInstance("n1")
9        .setLocation(Location.newInstance(-8,-4))
10       .addSizeDimension(STUFF, 20)
11       .setTimeWindow(TimeWindow.newInstance(0, 20000))
12       .setServiceTime(7).build();
13     Service n2 = Service.Builder.newInstance("n2")
14       .setLocation(Location.newInstance(-8,1))
15       .addSizeDimension(STUFF, 20)
16       .setTimeWindow(TimeWindow.newInstance(0, 20000))
17       .setServiceTime(7)
18       .build();
19     Service n3 = Service.Builder.newInstance("n3")
20       .setLocation(Location.newInstance(-5,4))
21       .addSizeDimension(STUFF, 50)
22       .setTimeWindow(TimeWindow.newInstance(0, 20000))
23       .setServiceTime(10)
24       .build();
25     Service n4 = Service.Builder.newInstance("n4")
26       .setLocation(Location.newInstance(-2,-7))
27       .addSizeDimension(STUFF, 25)
28       .setTimeWindow(TimeWindow.newInstance(0,20000))
29       .setServiceTime(10)
30       .build();
31     Service n5 = Service.Builder.newInstance("n5")
32       .setLocation(Location.newInstance(4,-4))
33       .addSizeDimension(STUFF, 25)
34       .setTimeWindow(TimeWindow.newInstance(0,20000))
35       .setServiceTime(10).build();
36     Service n6 = Service.Builder.newInstance("n6")
37       .setLocation(Location.newInstance(4,5))
38       .addSizeDimension(STUFF, 25)
39       .setTimeWindow(TimeWindow.newInstance(0,20000))
40       .setServiceTime(10)
41       .build();
42     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
43     costMatrixBuilder.addTransportTime(n0, n2.getLocation(),
44         roadFunction(n0, n2.getLocation()));
45     costMatrixBuilder.addTransportTime(n0, n1.getLocation(),
46         roadFunction(n0, n1.getLocation()));
47     costMatrixBuilder.addTransportTime(n1.getLocation(), n4.getLocation(),
48         roadFunction(n1.getLocation(), n4.getLocation()));
49     costMatrixBuilder.addTransportTime(n4.getLocation(), n5.getLocation(),
50         roadFunction(n4.getLocation(), n5.getLocation()));
51     costMatrixBuilder.addTransportTime(n5.getLocation(), n0,
52         roadFunction(n5.getLocation(), n0));
53     costMatrixBuilder.addTransportTime(n2.getLocation(), n3.getLocation(),
54         highwayFunction(n2.getLocation(), n3.getLocation()));
55     costMatrixBuilder.addTransportTime(n3.getLocation(), n6.getLocation(),
56         highwayFunction(n3.getLocation(), n6.getLocation()));
57     costMatrixBuilder.addTransportTime(n6.getLocation(), n0,
58         highwayFunction(n6.getLocation(), n0));
59     IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
60     vrpBuilder.setRoutingCost(cm);
61     VehicleType deliverType = VehicleTypeImpl.Builder.newInstance("vehicles00")
62       .addCapacityDimension(0, 180)
63       .build();
64     Vehicle vehicle = VehicleImpl.Builder.newInstance("vehicle")
65       .setType(deliverType).setStartLocation(n0).setLatestArrival(20000).build();
66     vrpBuilder.addVehicle(vehicle);
67     vrpBuilder.addAllJobs(Arrays.asList(n1, n4, n5));
68     VehicleRoutingProblem vrp = vrpBuilder.build();
69   }
70 }
```

Task: Q05JSAG (3/3)

Task (continuation)

```
71   public static double highwayFunction(Location end1, Location end2) {
72     return 2 * EuclideanDistanceCalculator
73       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 2;
74   }
75
76   public static double roadFunction(Location end1, Location end2) {
77     return 4 * EuclideanDistanceCalculator
78       .calculateDistance(end1.getCoordinate(), end2.getCoordinate())  + 4;
79   }
80 }
```

Correct solution

☑Line 67

Evaluation 2020, 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q06JSNW (1/7)

Introduction

In this task, you see a program that describes a network for a VRPTW. In addition, you see four more programs. Some of these programs describe the exact same network as the first program, even though they syntactically deviate from the first program. It is your task to find and report these equivalent programs.

Task

Look at the "Program to match" below. One or more of the four possible matches produce an equivalent network. Tick the respective boxes!

☐ Possible match 01	☐ Possible match 02	☐ Possible match 03	☐ Possible match 04

Program to match

```
1  public class Q06JSNW {
2
3    public static void main(String[] args) {
4      VehicleRoutingProblem.Builder vrpBuilder
5        = VehicleRoutingProblem.Builder.newInstance();
6
7      // Network
8      // --- Nodes
9      Location n0 = Location.newInstance(-2, -2);
10     Location n1 = Location.newInstance(-8, -4);
11     Location n2 = Location.newInstance(-8, -1);
12     Location n3 = Location.newInstance(-5, 4);
13     Location n4 = Location.newInstance(-2, -6);
14     Location n5 = Location.newInstance(2, 0);
15     Location n6 = Location.newInstance(4, 5);
16
17     // --- Edges
18     // --- --- roads
19     Set<RelationKey> roads = new HashSet<>();
20     roads.add(RelationKey.newKey(n0, n4));
21     roads.add(RelationKey.newKey(n0, n1));
22     roads.add(RelationKey.newKey(n0, n3));
23     roads.add(RelationKey.newKey(n4, n5));
24
25     // --- --- highways
26     Set<RelationKey> highways = new HashSet<>();
27     highways.add(RelationKey.newKey(n1, n2));
28     highways.add(RelationKey.newKey(n2, n3));
29     highways.add(RelationKey.newKey(n6, n5));
30
31     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
32
33     for(RelationKey key : roads)
34       costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
35
36     for(RelationKey key : highways)
37       costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
38
39     costMatrixBuilder.addTransportTime(n0,n5, specialFunction(n0, n5));
40     costMatrixBuilder.addTransportTime(n3, n6, specialFunction(n3,n6));
41
42     // adding to vrpBuilder etc. omitted
43   }
44
```

Task: Q06JSNW (2/7)

Task (continuation)

```
45   public static double highwayFunction(Location end1, Location end2) {
46     return 1.5 * EuclideanDistanceCalculator
47       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
48   }
49
50   public static double roadFunction(Location end1, Location end2) {
51     return 3 * EuclideanDistanceCalculator
52       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
53   }
54
55   public static double specialFunction(Location end1, Location end2) {
56     return 3 * EuclideanDistanceCalculator
57       .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
58   }
59   // RelationsKey class omitted for brevity
60 }
```

Task: Q06JSNW (3/7)

Task (continuation)

Matching option 1

```java
public class Q06JSNW_PossibleMatch1 {

  public static void main(String[] args) {
    VehicleRoutingProblem.Builder vrpBuilder
      = VehicleRoutingProblem.Builder.newInstance();

    // Network
    // --- Nodes
    Location n0 = Location.newInstance(-2, -2);
    Location n1 = Location.newInstance(-8, -4);
    Location n2 = Location.newInstance(-8, -1);
    Location n3 = Location.newInstance(-5, 4);
    Location n4 = Location.newInstance(-2, -6);
    Location n5 = Location.newInstance(2, 0);
    Location n6 = Location.newInstance(4, 5);

    IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();

    // --- Edges
    // --- --- highways
    Set<RelationKey> highways = new HashSet<>();
    highways.add(RelationKey.newKey(n1, n2));
    highways.add(RelationKey.newKey(n2, n3));
    highways.add(RelationKey.newKey(n6, n5));

    for(RelationKey key : highways)
      costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));

    costMatrixBuilder.addTransportTime(n0,n5, specialFunction(n0, n5));
    costMatrixBuilder.addTransportTime(n3, n6, specialFunction(n3, n6));

    // --- --- roads
    Set<RelationKey> roads = new HashSet<>();
    roads.add(RelationKey.newKey(n0, n4));
    roads.add(RelationKey.newKey(n0, n1));
    roads.add(RelationKey.newKey(n0, n3));
    roads.add(RelationKey.newKey(n4, n5));

    for(RelationKey key : roads)
       costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));

    // adding to vrpBuilder etc. omitted
  }

  public static double highwayFunction(Location end1, Location end2) {
    return 1.5 * EuclideanDistanceCalculator
      .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
  }

  public static double roadFunction(Location end1, Location end2) {
    return 3 * EuclideanDistanceCalculator
      .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
  }

  public static double specialFunction(Location end1, Location end2) {
    return 3 * EuclideanDistanceCalculator
      .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
  }
  // RelationsKey class omitted for brevity
}
```

Task: Q06JSNW (4/7)

Task (continuation)

Matching option 2

```
1  public class Q06JSNW_PossibleMatch2 {
2
3    public static void main(String[] args) {
4      VehicleRoutingProblem.Builder vrpBuilder
5        = VehicleRoutingProblem.Builder.newInstance();
6
7      // Network
8      // --- Nodes
9      Location n0 = Location.newInstance(-2, -2);
10     Location n1 = Location.newInstance(-8, -4);
11     Location n2 = Location.newInstance(-8, -1);
12     Location n3 = Location.newInstance(-5, 4);
13     Location n4 = Location.newInstance(-2, -6);
14     Location n5 = Location.newInstance(2, 0);
15     Location n6 = Location.newInstance(4, 5);
16
17     // --- Edges
18     // --- --- roads
19     Set<RelationKey> roads = new HashSet<>();
20     roads.add(RelationKey.newKey(n0,n4));
21     roads.add(RelationKey.newKey(n3, n6));
22     roads.add(RelationKey.newKey(n0, n3));
23     roads.add(RelationKey.newKey(n4, n5));
24
25     // --- --- highways
26     Set<RelationKey> highways = new HashSet<>();
27     highways.add(RelationKey.newKey(n1, n2));
28     highways.add(RelationKey.newKey(n2, n3));
29     highways.add(RelationKey.newKey(n6, n5));
30
31     // --- --- fastways
32     Set<RelationKey> fastways = new HashSet<>();
33     fastways.add(RelationKey.newKey(n0, n5));
34     fastways.add(RelationKey.newKey(n0, n1));
35
36     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
37
38     for(RelationKey key : roads)
39       costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
40
41     for(RelationKey key : highways)
42       costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
43
44     for(RelationKey key : fastways)
45       costMatrixBuilder.addTransportTime(key.from, key.to, specialFunction(key.from, key.to));
46
47     // adding to vrpBuilder etc. omitted
48   }
49
50   public static double highwayFunction(Location end1, Location end2) {
51     return 1.5 * EuclideanDistanceCalculator
52       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
53   }
54
55   public static double roadFunction(Location end1, Location end2) {
56     return 3 * EuclideanDistanceCalculator
57       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
58   }
59
60   public static double specialFunction(Location end1, Location end2) {
61     return 3 * EuclideanDistanceCalculator
62       .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
63   }
64
65   // RelationsKey class omitted for brevity
66
67 }
```

Task: Q06JSNW (5/7)

Task (continuation)

Matching option 3

```java
public class Q06JSNW_PossibleMatch3 {

  public static void main(String[] args) {
    VehicleRoutingProblem.Builder vrpBuilder
      = VehicleRoutingProblem.Builder.newInstance();

    // Network
    // --- Nodes
    Location n0 = Location.newInstance(-2, -2);
    Location n1 = Location.newInstance(-8, -4);
    Location n2 = Location.newInstance(-8, -1);
    Location n3 = Location.newInstance(-5, 4);
    Location n4 = Location.newInstance(-2, -6);
    Location n5 = Location.newInstance(2, 0);
    Location n6 = Location.newInstance(4, 5);

    // --- Edges
    // --- --- roads
    Set<RelationKey> roads = new HashSet<>();
    roads.add(RelationKey.newKey(n0, n4));
    roads.add(RelationKey.newKey(n0, n1));
    roads.add(RelationKey.newKey(n4, n5));

    // --- --- highways
    Set<RelationKey> highways = new HashSet<>();
    highways.add(RelationKey.newKey(n1, n2));
    highways.add(RelationKey.newKey(n3, n6));
    highways.add(RelationKey.newKey(n0, n3));
    highways.add(RelationKey.newKey(n6, n5));

    IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();

    for(RelationKey key : roads)
      costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));

    for(RelationKey key : highways)
      costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));

    costMatrixBuilder.addTransportTime(n0,n5, specialFunction(n0, n5));
    costMatrixBuilder.addTransportTime(n2, n3, specialFunction(n2, n3));

    // adding to vrpBuilder etc. omitted
  }

  public static double highwayFunction(Location end1, Location end2) {
    return 1.5 * EuclideanDistanceCalculator
      .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
  }

  public static double roadFunction(Location end1, Location end2) {
    return 3 * EuclideanDistanceCalculator
     .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
  }

  public static double specialFunction(Location end1, Location end2) {
    return 3 * EuclideanDistanceCalculator
      .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
  }

  // RelationsKey class omitted for brevity

}
```

Task: Q06JSNW (6/7)

Task (continuation)

Matching option 4

```java
public class Q06JSNW_PossibleMatch4 {

  public static void main(String[] args) {
    VehicleRoutingProblem.Builder vrpBuilder
      = VehicleRoutingProblem.Builder.newInstance();

    // Network
    // --- Nodes
    Location n0 = Location.newInstance(-2, -2);
    Location n1 = Location.newInstance(-8, -4);
    Location n2 = Location.newInstance(-8, -1);
    Location n3 = Location.newInstance(-5, 4);
    Location n4 = Location.newInstance(-2, -6);
    Location n5 = Location.newInstance(2, 0);
    Location n6 = Location.newInstance(4, 5);

    // --- Edges
    IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();

    costMatrixBuilder.addTransportTime(n0, n4, roadFunction(n0, n4));
    costMatrixBuilder.addTransportTime(n0, n5, specialFunction(n0, n5));
    costMatrixBuilder.addTransportTime(n0, n1, roadFunction(n0, n1));
    costMatrixBuilder.addTransportTime(n0, n3, roadFunction(n0, n3));
    costMatrixBuilder.addTransportTime(n4, n5, roadFunction(n4, n5));
    costMatrixBuilder.addTransportTime(n1, n2, highwayFunction(n1, n2));
    costMatrixBuilder.addTransportTime(n2, n3, highwayFunction(n2, n3));
    costMatrixBuilder.addTransportTime(n3, n6, specialFunction(n3, n6));
    costMatrixBuilder.addTransportTime(n6, n5, highwayFunction(n6, n5));

    // adding to vrpBuilder etc. omitted
  }

  public static double highwayFunction(Location end1, Location end2) {
    return 1.5 * EuclideanDistanceCalculator
      .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
  }

  public static double roadFunction(Location end1, Location end2) {
    return 3 * EuclideanDistanceCalculator
     .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
  }

  public static double specialFunction(Location end1, Location end2) {
    return 3 * EuclideanDistanceCalculator
      .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
  }

  // RelationsKey class omitted for brevity

}
```

Task: Q06JSNW (7/7)

Task (continuation)

Correct solution

☑Possible match 2, ☑Possible match 4

Evaluation 2020

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	10
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	10	MissingAnswAllowed	1 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Evaluation 2021

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	5 (default)
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	5	MissingAnswAllowed	1 (default)
WrgAnswersString	No (default)	MaxPointsToGet	10 (default)

Task: Q07JSALL (1/4)

Introduction

In this task, you first see a complete program. After that, you are shown excerpts from this program and you are asked to associate the correct semantics (meaning) with the language elements shown in the excerpt. Thus, it is your task to associate the correct semantics to pre-selected language elements.

Task

```
1  public class Q07JSNW {
2
3    public static int STUFF = 0;
4
5    public static void main(String[] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     Location n0 = Location.newInstance(-2, -2);
12
13     Service n1 = Service.Builder.newInstance("n1")
14     .setLocation(Location.newInstance(3, -4))
15     .addSizeDimension(STUFF, 20)
16     .setTimeWindow(TimeWindow.newInstance(10, 300))
17     .setServiceTime(7)
18     .build();
19
20     Service n2 = Service.Builder.newInstance("n2")
21     .setLocation(Location.newInstance(0, -3))
22     .addSizeDimension(STUFF, 20)
23     .setTimeWindow(TimeWindow.newInstance(20, 250))
24     .setServiceTime(7)
25     .build();
26
27     Service n3 = Service.Builder.newInstance("n3")
28     .setLocation(Location.newInstance(-5, 4))
29     .addSizeDimension(STUFF, 50)
30     .setTimeWindow(TimeWindow.newInstance(0, 140))
31     .setServiceTime(10)
32     .build();
33
34     Service n4 = Service.Builder.newInstance("n4")
35     .setLocation(Location.newInstance(-2, -6))
36     .addSizeDimension(STUFF, 25)
37     .setTimeWindow(TimeWindow.newInstance(0, 120))
38     .setServiceTime(10)
39     .build();
40
41     Service n5 = Service.Builder.newInstance("n5")
42     .setLocation(Location.newInstance(2, 0))
43     .addSizeDimension(STUFF, 25)
44     .setTimeWindow(TimeWindow.newInstance(0, 140))
45     .setServiceTime(10)
46     .build();
47
48     vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n4, n5));
49
50     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
51
52     // --- Edges
53     // --- --- roads
54     Set<RelationKey> roads = new HashSet<>();
55     roads.add(RelationKey.newKey(n0, n4));
56     roads.add(RelationKey.newKey(n0, n5));
57     roads.add(RelationKey.newKey(n0, n2));
58     roads.add(RelationKey.newKey(n0, n3));
59     roads.add(RelationKey.newKey(n1, n2));
60     roads.add(RelationKey.newKey(n1, n4));
61
```

Task: Q07JSALL (2/4)

Task (continuation)

```
62      for(RelationKey key : roads)
63        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
64
65      // --- --- highways
66      Set<RelationKey> highways = new HashSet<>();
67      highways.add(RelationKey.newKey(n3, n5));
68      highways.add(RelationKey.newKey(n2, n5));
69
70      for(RelationKey key : highways)
71        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
72
73      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
74      vrpBuilder.setRoutingCost(cm);
75
76      // Vehicle type and instance
77      VehicleType deliverType = VehicleTypeImpl.Builder.newInstance("vehicles00")
78        .addCapacityDimension(0, 180)
79        .build();
80
81      Vehicle vehicle = VehicleImpl.Builder.newInstance("vehicle")
82        .setType(deliverType).setStartLocation(n0).setLatestArrival(850).build();
83
84      vrpBuilder.addVehicle(vehicle);
85
86      VehicleRoutingProblem vrp = vrpBuilder.build();
87    }
88
89    public static double highwayFunction(Location end1, Location end2) {
90      return 1.2 * EuclideanDistanceCalculator
91        .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 1.3;
92    }
93
94    public static double roadFunction(Location end1, Location end2) {
95      return 2.5 * EuclideanDistanceCalculator
96        .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3.2;
97    }
98  }
```

Task: Q07JSALL (3/4)

Task (continuation)

1st Element[a]

```
89 public static double highwayFunction(Location end1, Location end2) {
90       return 1.2 * EuclideanDistanceCalculator
91          .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 1.3;
92 }
```

Which description of the sematics and usage of the the highwayFunction() method in the program above is the most appropriate?

- ○ It is used for single customers (Services). Together with an expression that refers to the respective customer it determines how long the time window of this customer is opened
- ○ It is used for single vehicles. Together with an expression it determines the costs that occur upon deployment of the respective vehicle.
- ○ It is used for single edges, that is the connection between two customers. Together with an expression it determines how long it takes a vehicle to travel the respective edge.
- ○ It is used for single depots. Together with an expression it determines the capacity of the vehicles starting from that depot.

2nd Element[b]

```
65 // --- --- highways
66 Set<RelationKey> highways = new HashSet<>();
67 highways.add(RelationKey.newKey(n3, n5));
68 highways.add(RelationKey.newKey(n2, n5));
69
70 for(RelationKey key : highways)
71    costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
```

Which explanation concerning the meaning of the depicted highways set in combination with the ensuing loop in the context of the complete program is most appropriate?

- ○ It allows to group vehicles that will then jointly travel the respective edge.
- ○ All edges (connections between two locations) must be associated with a group because only via a group it is possible to set the duration function of an edge.
- ○ All Service instances that are connected by edges of the same group are merged to one single Service so that the vehicle only needs to visit one of these customers.
- ○ They are an optional program construct that may facillitate the definition of edges. Especially in cases in which a large number of edges are assigned the same (duration) function.

[a] Element missing in actual survey.
[b] Element missing in actual survey.

Task: Q07JSALL (4/4)

Task (continuation)

Correct solution

1st Element

- It is used for single edges, that is the connection between two customers. Together with an expression it determines how long it takes a vehicle to travel the respective edge.

2nd Element

- They are an optional program construct that may facillitate the definition of edges. Especially in cases in which a large number of edges are assigned the same (duration) function.

Evaluation 2020, 2021

	Parameter	Value	Parameter	Value
1st Element	Answer Cell	Yes	NegPointsP	5 (default)
	CorAnswersString	Yes	WrongAnswAllowed	0 (default)
	Points	5	MissingAnswAllowed	0 (default)
	WrgAnswersString	No (default)	MaxPointsToGet	5 (default)
	Parameter	Value	Parameter	Value
2nd Element	Answer Cell	Yes	NegPointsP	5 (default)
	CorAnswersString	Yes	WrongAnswAllowed	0 (default)
	Points	5	MissingAnswAllowed	0 (default)
	WrgAnswersString	No (default)	MaxPointsToGet	5 (default)

Task: Q08JSALL (1/4)

Introduction

In this task, you first see a complete program. The program features some comments. These comments represent TODOs, i.e. future programming tasks. In addition, you'll find some questions that ask for the elements, that will be affected by these TODOs. Your task is to answer these questions correctly.

Task

```
1  public class Q08JSALL2ND {
2    public static int SOAP = 0;
3    public static int PAPER = 1;
4
5    public static void main(String[] args) {
6      Examples.createOutputFolder();
7
8      VehicleRoutingProblem.Builder vrpBuilder
9        = VehicleRoutingProblem.Builder.newInstance();
10
11     // Network
12     // --- Nodes
13     /* TODO: The customer with the shortest service time must be added to the
14     * respective depot */
15     Location n0 = Location.newInstance(2, 4);
16     Location n1 = Location.newInstance(-5, -2);
17     Location n2 = Location.newInstance(0, -3);
18     Location n3 = Location.newInstance(-5, 4);
19     Location n4 = Location.newInstance(-2, -2);
20     Location n5 = Location.newInstance(2, 0);
21     Location n6 = Location.newInstance(-2, -6);
22     Location n7 = Location.newInstance(3, -4);
23
24     Delivery dn1soap = Delivery.Builder.newInstance("dn1soap")
25       .setLocation(n1)
26       .addSizeDimension(SOAP, 10)
27       .setTimeWindow(TimeWindow.newInstance(20,220))
28       .setServiceTime(8)
29       .build();
30
31     Delivery dn1paper = Delivery.Builder.newInstance("dn1paper")
32       .setLocation(n1)
33       .addSizeDimension(PAPER, 20)
34       .setTimeWindow(TimeWindow.newInstance(20,220))
35       .setServiceTime(8)
36       .build();
37
38     Delivery dn2soap = Delivery.Builder.newInstance("dn2soap")
39       .setLocation(n2)
40       .addSizeDimension(SOAP, 20)
41       .setTimeWindow(TimeWindow.newInstance(20, 250))
42       .setServiceTime(2)
43       .build();
44
45     Delivery dn3soap = Delivery.Builder.newInstance("dn3soap")
46       .setLocation(n3)
47       .addSizeDimension(SOAP, 50)
48       .setTimeWindow(TimeWindow.newInstance(0, 140))
49       .setServiceTime(30)
50       .build();
51
52     Delivery dn4soap = Delivery.Builder.newInstance("dn4soap")
53       .setLocation(n4)
54       .addSizeDimension(SOAP, 60)
55       .setTimeWindow(TimeWindow.newInstance(0, 150))
56       .setServiceTime(10)
57       .build();
58
```

Task: Q08JSALL (2/4)

Task (continuation)

```
59      Delivery dn4paper = Delivery.Builder.newInstance("dn4paper")
60        .setLocation(n4)
61        .addSizeDimension(PAPER, 70)
62        .setTimeWindow(TimeWindow.newInstance(0, 150))
63        .setServiceTime(10)
64        .build();
65
66      Delivery dn5soap = Delivery.Builder.newInstance("dn5soap")
67        .setLocation(n5)
68        .addSizeDimension(SOAP, 30)
69        .setTimeWindow(TimeWindow.newInstance(0, 140))
70        .setServiceTime(10)
71        .build();
72
73      Delivery dn5paper = Delivery.Builder.newInstance("dn5paper")
74        .setLocation(n5)
75        .addSizeDimension(PAPER, 40)
76        .setTimeWindow(TimeWindow.newInstance(0, 140))
77        .setServiceTime(10)
78        .build();
79
80      Delivery dn6soap = Delivery.Builder.newInstance("dn6soap")
81        .setLocation(n6)
82        .addSizeDimension(SOAP, 25)
83        .setTimeWindow(TimeWindow.newInstance(0, 120))
84        .setServiceTime(10)
85        .build();
86
87      Delivery dn7soap = Delivery.Builder.newInstance("dn7soap")
88        .setLocation(n7)
89        .addSizeDimension(SOAP, 20)
90        .setTimeWindow(TimeWindow.newInstance(10, 300))
91        .setServiceTime(7)
92        .build();
93
94      Location n8 = Location.newInstance(-5, -6);
95
96      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
97
98      // --- Edges
99      // --- --- roads
100     Set<RelationKey> roads = new HashSet<>();
101     roads.add(RelationKey.newKey(n0, n6));
102     roads.add(RelationKey.newKey(n0, n5));
103     roads.add(RelationKey.newKey(n3, n4));
104     roads.add(RelationKey.newKey(n0, n3));
105     roads.add(RelationKey.newKey(n7, n2));
106     roads.add(RelationKey.newKey(n7, n6));
107     roads.add(RelationKey.newKey(n3, n0));
108     roads.add(RelationKey.newKey(n5, n0));
109     roads.add(RelationKey.newKey(n3, n1));
110     roads.add(RelationKey.newKey(n1, n8));
111     roads.add(RelationKey.newKey(n4, n2));
112     roads.add(RelationKey.newKey(n8, n6));
113
114     for(RelationKey key : roads)
115       costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
116
117     // --- --- highways
118     /* TODO: two highways will later be modelled here:
119     * one going from the soap depot to the customer with the highest demands in both soap
120     * and paper and the other going from the customer with the highest soap and
121     * paper demands to the paper depot.    */
122     Set<RelationKey> highways = new HashSet<>();
123     highways.add(RelationKey.newKey(n3, n5));
124     highways.add(RelationKey.newKey(n2, n5));
125
126     for(RelationKey key : highways)
127       costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
128
```

Task: Q08JSALL (3/4)

Task (continuation)

```
129     IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
130     vrpBuilder.setRoutingCost(cm);
131
132     // Vehicle type and instance
133
134     VehicleType soapVehicle = VehicleTypeImpl.Builder.newInstance("soapVehicle")
135       .addCapacityDimension(SOAP, 180).build();
136
137     VehicleType paperVehicle = VehicleTypeImpl.Builder.newInstance("paperVehicle")
138       .addCapacityDimension(PAPER, 180).build();
139
140     VehicleImpl soapVehicleInstance = VehicleImpl.Builder.newInstance("soapVehicleInstance")
141       .setType(soapVehicle)
142       .setStartLocation(n0)
143       .build();
144
145     VehicleImpl paperVehicleInstance = VehicleImpl.Builder.newInstance("paperVehicleInstance")
146       .setType(paperVehicle)
147       .setStartLocation(n8)
148       .build();
149
150     vrpBuilder.addAllJobs(Arrays.asList(dn1paper, dn1soap, dn2soap, dn3soap, dn4paper, dn4soap,
151                                         dn5paper, dn5soap, dn6soap, dn7soap));
152     vrpBuilder.addVehicle(soapVehicleInstance);
153     vrpBuilder.addVehicle(paperVehicleInstance);
154
155     VehicleRoutingProblem vrp = vrpBuilder.build();
156   }
157 }
```

1. Which customer must be added?

Read the comment in lines 13 and 14. Which customer must be added to the respective depot according to the comment?

Note: Some of the provided answers might not be customers.

○ n0	○ n1	○ n2
○ n3	○ n4	○ n5
○ n6	○ n7	○ n8

2. Which is the correct depot?

Read the comment in lines 13 and 14 . To which depot must the customer be added to?

Note: Some of the provided answers might not be depots.

○ n0	○ n1	○ n2
○ n3	○ n4	○ n5
○ n6	○ n7	○ n8

3. Which nodes are affected?

Read the comment that spans from lines 118 to line 121. What are the three nodes this comment refers to?

Note: Some of the provided answers might not be customers.

☐ n0	☐ n1	☐ n2
☐ n3	☐ n4	☐ n5
☐ n6	☐ n7	☐ n8

Task: Q08JSALL (4/4)

Task (continuation)

Correct solution

1.	◉ n2
2.	◉ n0
3.	☑ n0, ☑ n4, ☑ n8

Evaluation 2020, 2021

1.

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	2
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	2	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	2

2.

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	2
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	2	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	2

3.

Parameter	Value	Parameter	Value
Answer Cell	Yes	NegPointsP	2
CorAnswersString	Yes	WrongAnswAllowed	0 (default)
Points	2	MissingAnswAllowed	0 (default)
WrgAnswersString	No (default)	MaxPointsToGet	6

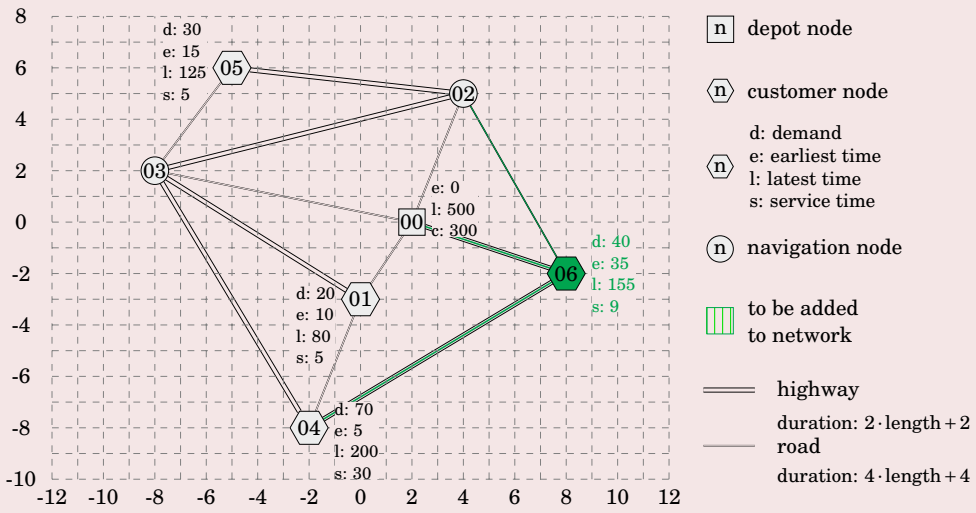
Task: Q09JSNW (1/4)

Introduction

In this task you'll find a graphical representation of a network. The network comprises a set of customers with demands, time windows and service times. Further below is the corresponding program. However, the program still shows some gaps. The gaps can also be seen in the illustration: they correspond to the elements drawn using green color. Your task is to fill in the gaps so that the program describes the complete network.

Task

Explanation: Below is the illustration of a network. The network comprises a set of customers with demands, time windows and service times.
Further below is the corresponding JSprit model. However, the model still has some gaps. The gaps can also be seen in the illustration: they correspond to the elements drawn using green color.
Fill in the gaps so that the JSprit model represents the complete network.




```
1  public class Q09JSNW {
2    public static int STUFF = 0;
3
4    public static void main(String[] args) {
5
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     Location n0 = Location.newInstance(2, 0);
12
13     Service n1 = Service.Builder.newInstance("n1")
14       .setLocation(Location.newInstance(0,-3))
15       .addSizeDimension(STUFF, 20)
16       .setTimeWindow(TimeWindow.newInstance(10, 80))
17       .setServiceTime(5)
18       .build();
19
20     Location n2 = Location.newInstance(4,5);
21
22     Location n3 = Location.newInstance(-8, 2);
23
```

Task: Q09JSNW (2/4)

Task (continuation)

```
24      Service n4 = Service.Builder.newInstance("n4")
25        .setLocation(Location.newInstance(-2, -8))
26        .addSizeDimension(STUFF, 70)
27        .setTimeWindow(TimeWindow.newInstance(5, 200))
28        .setServiceTime(30)
29        .build();
30
31      Service n5 = Service.Builder.newInstance("n5")
32        .setLocation(Location.newInstance(-5, 6))
33        .addSizeDimension(STUFF, 30)
34        .setTimeWindow(TimeWindow.newInstance(15, 125))
35        .setServiceTime(5)
36        .build();
37
38      //TASK 1.1: SOME TEXT TO BE ADDED HERE
39
40      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
41
42      // --- Edges
43      // --- --- roads
44      Set<RelationKey> roads = new HashSet<>();
45      roads.add(RelationKey.newKey(n0, n2));
46      roads.add(RelationKey.newKey(n0, n3));
47      roads.add(RelationKey.newKey(n0, n1));
48      roads.add(RelationKey.newKey(n1, n4));
49      roads.add(RelationKey.newKey(n3, n5));
50      // TASK 2: SOME TEXT TO BE ADDED HERE
51
52      for(RelationKey key : roads)
53        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
54
55      // --- --- highways
56      Set<RelationKey> highways = new HashSet<>();
57      highways.add(RelationKey.newKey(n2, n5));
58      highways.add(RelationKey.newKey(n2, n3));
59      highways.add(RelationKey.newKey(n1, n3));
60      highways.add(RelationKey.newKey(n3, n4));
61      // TASK 3: SOME TEXT TO BE ADDED HERE
62
63      for(RelationKey key : highways)
64        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
65
66      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
67      vrpBuilder.setRoutingCost(cm);
68
69      // Vehicle type and instance
70
71      VehicleType vehicles = VehicleTypeImpl.Builder.newInstance("vehicles")
72        .addCapacityDimension(STUFF, 300)
73        .build();
74
75      VehicleImpl vehiclesInstance = VehicleImpl.Builder.newInstance("soapVehicleInstance")
76        .setType(vehicles)
77        .setStartLocation(n0)
78        .setLatestArrival(300)
79        .build();
80
81      vrpBuilder.addAllJobs(Arrays.asList(n1, n4, n5 /* TASK 1.2: SOME TEXT TO BE ADDED HERE */ ));
82
83      vrpBuilder.addVehicle(vehiclesInstance);
84
85      VehicleRoutingProblem vrp = vrpBuilder.build();
86
87    }
88
```

Task: Q09JSNW (3/4)

Task (continuation)

```
89   public static double highwayFunction(Location end1, Location end2) {
90     return 2 * EuclideanDistanceCalculator
91       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 2;
92   }
93
94   public static double roadFunction(Location end1, Location end2) {
95     return 4 * EuclideanDistanceCalculator
96       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 4;
97   }
98
99   // static class RelationKey (omitted for brevity)
100 }
```

Task 1: In the gap program above, you find comments that read "TASK 1.1: TEXT TO BE ADDED HERE" and "TASK 1.2: TEXT TO BE ADDED HERE". Both comments mark places in the program where you need to insert additional text (code) in order for the program to comply with the network given in the illustration above. In the following two text areas, enter the code (one or more lines) that should replace these comments in order to complete the program.
Note: If necessary, please replace '>' with '>' and '<' with ' <' (or use ‹ ›)

Task 1.1: Enter answer here

Task 1.2: Enter answer here

Task 2: In the following text area, enter the code (one or more lines) that should replace the comment " TASK 2: SOME TEXT TO BE ADDED HERE " in the complete program above.
Note: If necessary, please replace '>' with '>' and '<' with ' <' (or use ‹ ›)

Enter answer here

Task 3: In the following text area, enter the code (one or more lines) that should replace the comment " TASK 3: SOME TEXT TO BE ADDED HERE " in the complete program above.
Note: If necessary, please replace '>' with '>' and '<' with ' <' (or use ‹ ›)

Enter answer here

Task: Q09JSNW (4/4)

Task (continuation)

Correct solution

Task 1.1:

```
38 Service n6 = Service.Builder.newInstance("n6")
39              .setLocation(Location.newInstance(8, -2))
40              .addSizeDimension(STUFF, 40)
41              .setTimeWindow(TimeWindow.newInstance(35, 155))
42              .setServiceTime(9)
43              .build();
```

Task 1.2:

```
81 vrpBuilder.addAllJobs(Arrays.asList(n1, n4, n5, n6 ));
```

Task 2:

```
50 roads.add(RelationKey.newKey(n2, n6));
```

Task 3:

```
61 highways.add(RelationKey.newKey(n0, n6));
62 highways.add(RelationKey.newKey(n4, n6));
```

Evaluation 2020, 2021

Task 1: 6 Points.
Task 2: 2 Points.
Task 3: 2 Points.

Task: Q09JSAG (1/5)

Introduction

In this task, you'll find an incomplete program together with a visual representation of the network represented by this program. Complete the program in a way so that the target network and agent behaviour visualised at the bottom of the page result. Note that the modelled behaviour, i.e. the tour, must be a likely outcome of the agent behaviour that you modelled in the program.

Note: In this task, there are some nodes for which demands, time windows, and service times are to be defined even though the vehicle must not service them! Navigation nodes (as well as those customer not supposed to be serviced) can be visited by the agent (vehicle) but they will not receive a delivery. Find a way to program this.

Task

```
1  public class Q09JSAG {
2    public static int STUFF = 0;
3
4    public static void main(String[] args) {
5
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      /* TASK 1 BEGINNING */
10     // Network
11     // --- Nodes
12     Location l0 = Location.newInstance(2, 4);
13     Location l1 = Location.newInstance(-2, -8);
14     Location l2 = Location.newInstance(8, -6);
15     Location l3 = Location.newInstance(-10, -6);
16     Location l4 = Location.newInstance(-4, -2);
17     Location l5 = Location.newInstance(-5, 6);
18     Location l6 = Location.newInstance(7, 4);
19
20     Service n1 = Service.Builder.newInstance("n1")
21       .setLocation(l1)
22       .addSizeDimension(0, 30)
23       .setTimeWindow(TimeWindow.newInstance(10, 90))
24       .setServiceTime(5)
25       .build();
26
27     vrpBuilder.addAllJobs(Arrays.asList(n1));
28     /* TASK 1 END */
29
30     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
31
32     // --- roads
33     Set<RelationKey> roads = new HashSet<>();
34     roads.add(RelationKey.newKey(l0, l1));
35     roads.add(RelationKey.newKey(l0, l2));
36     roads.add(RelationKey.newKey(l0, l6));
37     roads.add(RelationKey.newKey(l0, l4));
38     roads.add(RelationKey.newKey(l4, l5));
39     roads.add(RelationKey.newKey(l1, l2));
40
41     for(RelationKey key : roads)
42       costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
43
44     // --- highways
45     Set<RelationKey> highways = new HashSet<>();
46     highways.add(RelationKey.newKey(l2, l6));
47     highways.add(RelationKey.newKey(l6, l5));
48     highways.add(RelationKey.newKey(l3, l5));
49     highways.add(RelationKey.newKey(l1, l3));
50     highways.add(RelationKey.newKey(l3, l4));
51
52     for(RelationKey key : highways)
53       costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
54
55     IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
56     vrpBuilder.setRoutingCost(cm);
57
```

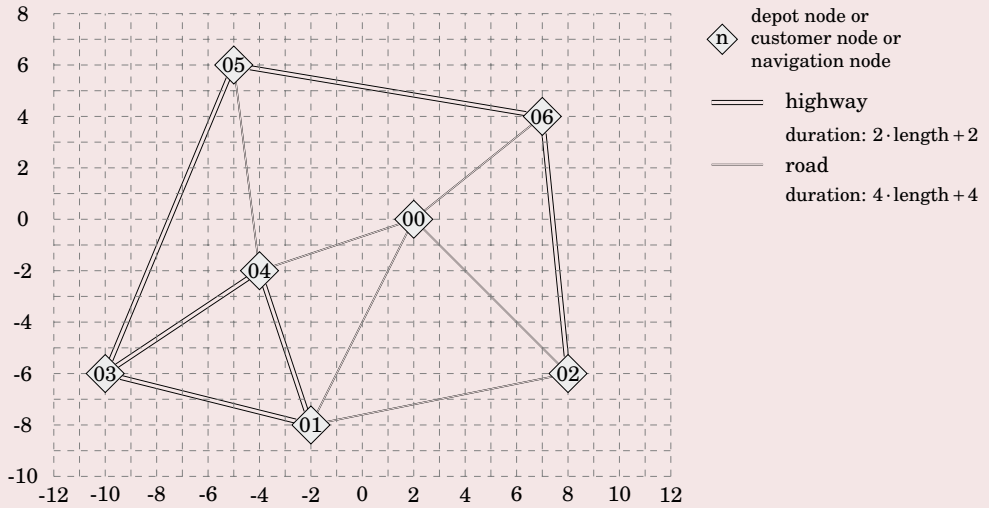
Task: Q09JSAG (2/5)

Task (continuation)

Task

```
58      /* TASK 2 BEGINNING */
59      // Vehicle type definition
60      VehicleType stuffVehicleType = VehicleTypeImpl.Builder.newInstance("stuffVehicleType")
61        .build();
62
63      // Vehicle instance defintion
64      VehicleImpl stuffVehicleInstance = VehicleImpl.Builder.newInstance("stuffVehicleInstance")
65        .setType(stuffVehicleType)
66        .setStartLocation(l0)
67        .build();
68      // Adding vehicle instance to the problem
69      vrpBuilder.addVehicle(stuffVehicleInstance);
70      /* TASK 2 END */
71
72      VehicleRoutingProblem vrp = vrpBuilder.build();
73    }
74    public static double highwayFunction(Location end1, Location end2) {
75      return 2 * EuclideanDistanceCalculator
76      .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 2;
77    }
78
79    public static double roadFunction(Location end1, Location end2) {
80      return 4 * EuclideanDistanceCalculator
81        .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 4;
82    }
83
84    //-- static class RelationKey omitted for brevity
85  }
```

Current state

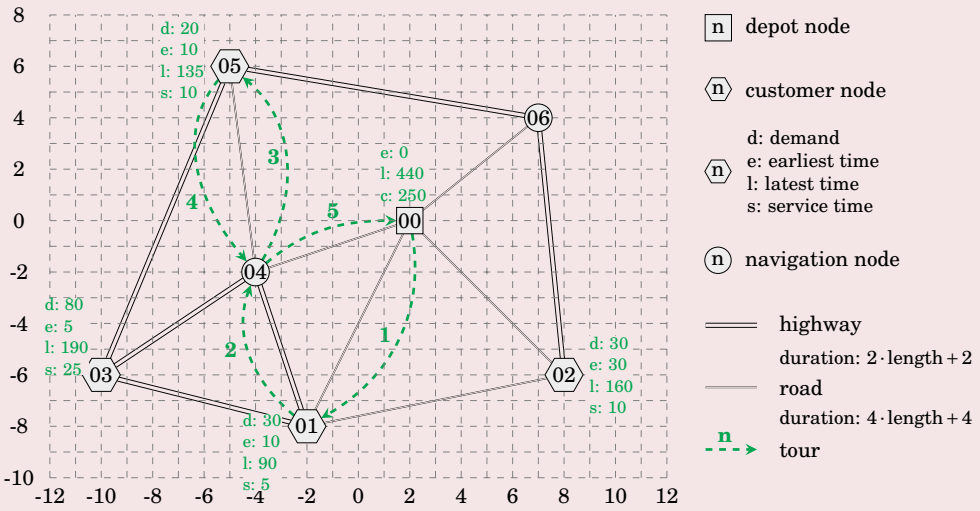


Task: Q09JSAG (3/5)

Task (continuation)

Task

Target state



Task 1: From the listing above, copy and paste the code betweenthe comment "TASK 1 : BEGINNING" and the comment "TASK 1 END". Complete the copied code for the network (including depot and customer definitions) so that it conforms to the target state depicted in the illustration.
Note: If necessary, please replace '>' with '>' and '<' with ' <' (or use ‹ ›)

Enter answer here

Task 2: From the listing above, copy and paste the code betweenthe comment "TASK 2 : BEGINNING" and the comment "TASK 2 END". Complete the copied code for the network (including depot and customer definitions) so that it conforms to the target state depicted in the illustration. Hint: Take a good look! Something is missing!
Note: If necessary, please replace '>' with '>' and '<' with ' <' (or use ‹ ›)

Enter answer here

Task: Q09JSAG (4/5)

Task (continuation)

Correct solution

Task 1:

```
//Network
      // --- Nodes
      Location l0 = Location.newInstance(2, 4);
      Location l1 = Location.newInstance(-2, -8);
      Location l2 = Location.newInstance(8, -6);
      Location l3 = Location.newInstance(-10, -6);
      Location l4 = Location.newInstance(-4, -2);
      Location l5 = Location.newInstance(-5, 6);
      Location l6 = Location.newInstance(7, 4);

      Service n1 = Service.Builder.newInstance("n1")
            .setLocation(l1)
            .addSizeDimension(0, 30)
            .setTimeWindow(TimeWindow.newInstance(10, 90))
            .setServiceTime(5)
            .build();

      Service n2 = Service.Builder.newInstance("n2")
                  .setLocation(l2)
                  .addSizeDimension(0, 30)
                  .setTimeWindow(TimeWindow.newInstance(30, 160))
                  .setServiceTime(10)
                  .build();

      Service n3 = Service.Builder.newInstance("n3")
                  .setLocation(l3)
                  .addSizeDimension(0, 80)
                  .setTimeWindow(TimeWindow.newInstance(5, 190))
                  .setServiceTime(25)
                  .build();

      Service n5 = Service.Builder.newInstance("n5")
                  .setLocation(l5)
                  .addSizeDimension(0, 20)
                  .setTimeWindow(TimeWindow.newInstance(10, 135))
                  .setServiceTime(10)
                  .build();

      vrpBuilder.addAllJobs(Arrays.asList(n1,n5));
```

Task 2:

```
    // Vehicle type definition
    VehicleType stuffVehicleType = VehicleTypeImpl.Builder.newInstance("stuffVehicleType")
          .addCapacityDimensions(STUFF, 50)
          .build();

    // Vehicle instance defintion
    VehicleImpl stuffVehicleInstance = VehicleImpl.Builder.newInstance("stuffVehicleInstance")
          .setType(stuffVehicleType)
          .setStartLocation(l0)
          .build();
    // Adding vehicle instance to the problem
    vrpBuilder.addVehicle(stuffVehicleInstance);
```

Evaluation 2020

Task 1: 10 Points.
Task 2: Not evaluated

Task: Q09JSAG (5/5)

Evaluation 2021

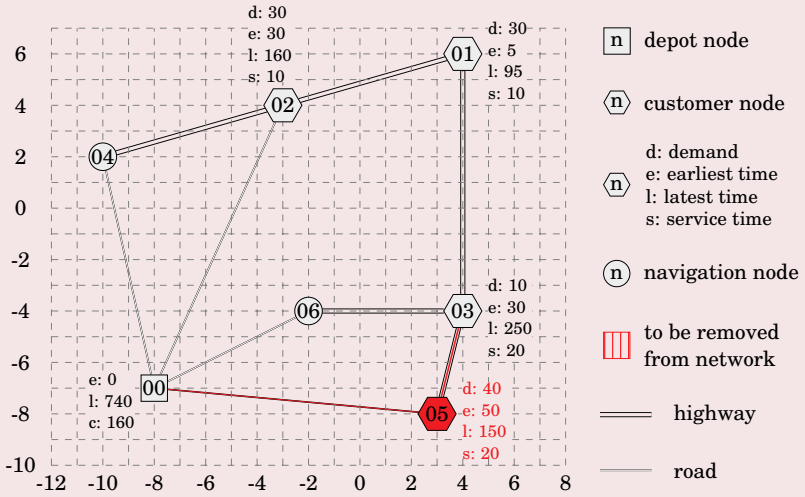
Task 1: 7 Points. Syntactical mistakes -1P; missing demand definitions (all) -4 P.; missing demand definitions (one correctly present) -2 P.; missing lines of code -1 P.; wrong value for demand or time -1 P. (max -2 P.); wrong or missing customer declaration -2 P.; wrong code (no own code additions) -8 P.; wrong customer label (specified in constructor) -1 P.; superflous code -1 P.; mixed-up customer (mistaken customer x for y) -1 P., wrong or incompatible asssigment (-2 P.); wrong location -1 P.; unecessary changes to pre-defined customer (introduction of mistakes) -1 P., missing call of build() method -1 P.
Task 2: 3 Points. Missing capacity dimension -2 P., Wrong product (only 'STUFF' and '0' are correct) -1P, Wrong value for either capacity dimension or latest time -1 P. (-2 P. max), missing latest time -2 P.

Task: Q10JSNW (1/3)

Introduction

In this task, you will find an illustration of a network comprised of highways, roads, navigation and customer nodes. In addition, you will find a program that corresponds to this illustration.In the illustration, some elements (e.g. nodes, demands, highways, etc.) are drawn in red color. These are the elements that are to be removed from the program.
Note: At the bottom of the page is a text area. Copy and paste the code that corresponds to the elements that need to be deleted into this text area. The order in which you paste the elements is not important.

Task




```
1  public class Q10JSNW {
2
3    public static int STUFF = 0;
4
5    public static void main(String[]] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      Location n0 = Location.newInstance(-8, -7);
10
11     Service n1 = Service.Builder.newInstance("n1")
12       .setLocation(Location.newInstance(4, 6))
13       .addSizeDimension(STUFF, 30)
14       .setTimeWindow(TimeWindow.newInstance(5, 95))
15       .setServiceTime(10)
16       .build();
17
18     Service n2 = Service.Builder.newInstance("n2")
19       .setLocation(Location.newInstance(-3, 4))
20       .addSizeDimension(STUFF, 30)
21       .setTimeWindow(TimeWindow.newInstance(5, 95))
22       .setServiceTime(10)
23       .build();
24
25     Service n3 = Service.Builder.newInstance("n3")
26       .setLocation(Location.newInstance(4, -4))
27       .addSizeDimension(STUFF, 10)
28       .setTimeWindow(TimeWindow.newInstance(30, 250))
29       .setServiceTime(20)
30       .build();
31
32     Location n4 = Location.newInstance(-10, 2);
33
```

Task: Q10JSNW (2/3)

Task (continuation)

```
60      Service n5 = Service.Builder.newInstance("n5")
61        .setLocation(Location.newInstance(3, -8))
62        .addSizeDimension(STUFF, 40)
63        .setTimeWindow(TimeWindow.newInstance(50, 150))
64        .setServiceTime(20)
65        .build();
66
67      Location n6 = Location.newInstance(-2, -4);
68
69      IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
70
71      // --- roads
72      Set<RelationKey> roads = new HashSet<>();
73      roads.add(RelationKey.newKey(n0, n4));
74      roads.add(RelationKey.newKey(n0, n2));
75      roads.add(RelationKey.newKey(n0, n6));
76      roads.add(RelationKey.newKey(n0, n5));
77
78      for(RelationKey key : roads)
79        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
80
81      // --- highways
82      Set<RelationKey> highways = new HashSet<>();
83      highways.add(RelationKey.newKey(n2, n4));
84      highways.add(RelationKey.newKey(n1, n2));
85      highways.add(RelationKey.newKey(n1, n3));
86      highways.add(RelationKey.newKey(n3, n5));
87      highways.add(RelationKey.newKey(n3, n6));
88
89      for(RelationKey key : highways)
90        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
91
92      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
93      vrpBuilder.setRoutingCost(cm);
94
95      VehicleType stuffVehicleType = VehicleTypeImpl.Builder.newInstance("stuffVehicleType")
96        .addCapacityDimension(STUFF, 160).build();
97
98      VehicleImpl stuffVehicleInstance = VehicleImpl.Builder.newInstance("stuffVehicleInstance")
99        .setType(stuffVehicleType)
100       .setLatestArrival(740)
101       .setStartLocation(n0)
102       .build();
103     // Adding vehicle instance to the problem
104     vrpBuilder.addVehicle(stuffVehicleInstance);
105
106     VehicleRoutingProblem vrp = vrpBuilder.build();
107   }
108
109   public static double highwayFunction(Location end1, Location end2) {
110     return 1.5 * EuclideanDistanceCalculator
111       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
112   }
113
114   public static double roadFunction(Location end1, Location end2) {
115     return 3 * EuclideanDistanceCalculator
116       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
117   }
118
119 }
```

Task: Q10JSNW (3/3)

Task (continuation)

From above JSprit program, copy those lines that need to be deleted and paste them in the following text area (in an arbitrary order).
Note: If necessary, please replace '>' with '>' and '<' with ' <' (or use ‹ ›)

Enter answer here

Correct solution

```
60 Service n5 = Service.Builder.newInstance("n5")
61              .setLocation(Location.newInstance(3, -8))
62              .addSizeDimension(STUFF, 40)
63              .setTimeWindow(TimeWindow.newInstance(50, 150))
64              .setServiceTime(20)
65              .build();
76 roads.add(RelationKey.newKey(n0, n5));
```

Evaluation 2020, 2021

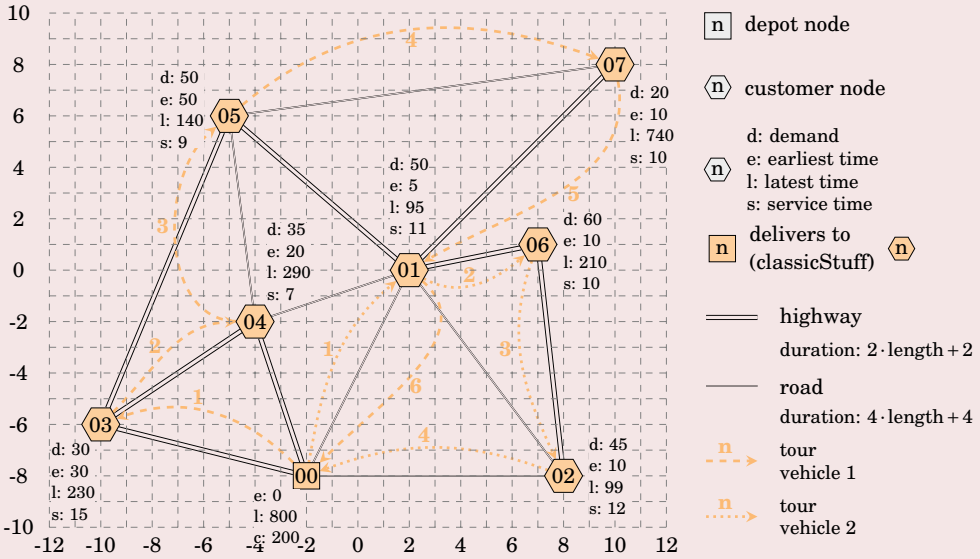
Scheme: 10 Points. Missing removal: -5 Points.

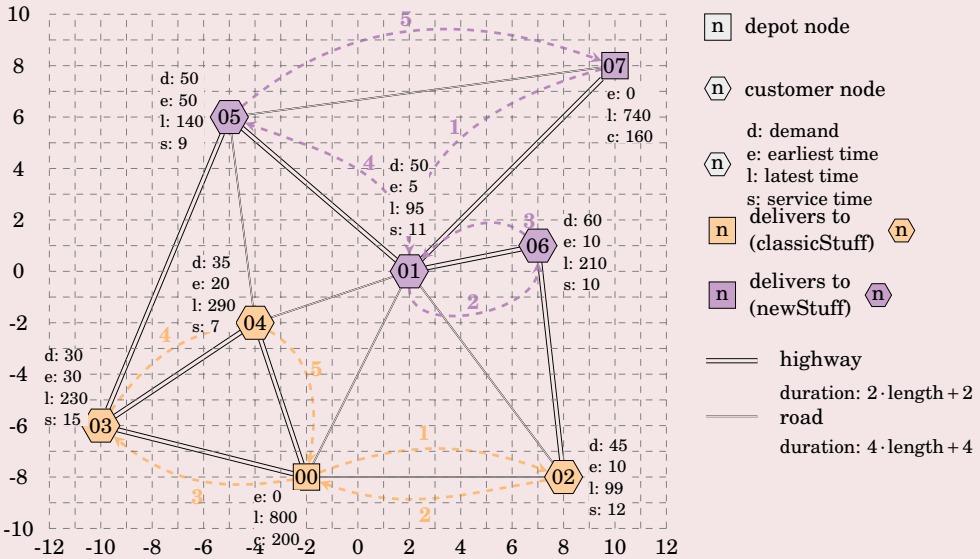
Task: Q11JSALL (1/5)

Introduction

In this task, you will find two graphical representations of networks that are comprised of highways, roads, navigation nodes and customer nodes.
In the first graphical representation you will find one depot together with **seven customer nodes.** The customer nodes are visited by two different tours.
After the two graphical network representations, you will find a program that corresponds **to the first graphical network representation** (i.e. it describes this representation).
The second graphical network representation displays the target state in which you are to transform the program: One of the customer nodes was transformed into a depot from which a new product is delivered to some of the customers. These customers only have a demand for the new product and do no longer require the old one (in other words, they are only supplied by one depot).
The program features comments that mark the beginning and the end of program sections that must be modified in order to transform the program into the target state. At the end of the page, there are corresponding text areas, in which you copy, paste and modify the original code in a suitable way.

Task





Task: Q11JSALL (2/5)

Task (continuation)

```
1  public class Q11JSALL {
2    public static final int CLASSIC_STUFF = 0;
3    public static final int NEW_STUFF = 1;
4
5    public static void main(String[] args) {
6      VehicleRoutingProblem.Builder vrpBuilder
7        = VehicleRoutingProblem.Builder.newInstance();
8
9      // Network
10     // --- Nodes
11     /* TASK 1 BEGINNING */
12     Location n0 = Location.newInstance(-2, -8);
13
14     Delivery n1 = Delivery.Builder.newInstance("n1")
15       .setLocation(Location.newInstance(2,0))
16       .addSizeDimension(CLASSIC_STUFF, 50)
17       .setTimeWindow(TimeWindow.newInstance(5, 95))
18       .setServiceTime(11)
19       .build();
20     Delivery n2 = Delivery.Builder.newInstance("n2")
21       .setLocation(Location.newInstance(8,-8))
22       .addSizeDimension(CLASSIC_STUFF, 45)
23       .setTimeWindow(TimeWindow.newInstance(10, 99))
24       .setServiceTime(12)
25       .build();
26     Delivery n3 = Delivery.Builder.newInstance("n3")
27       .setLocation(Location.newInstance(-10,-6))
28       .addSizeDimension(CLASSIC_STUFF, 40)
29       .setTimeWindow(TimeWindow.newInstance(10, 99))
30       .setServiceTime(12)
31       .build();
32     Delivery n4 = Delivery.Builder.newInstance("n4")
33       .setLocation(Location.newInstance(-4, -2))
34       .addSizeDimension(CLASSIC_STUFF, 35)
35       .setTimeWindow(TimeWindow.newInstance(20, 290))
36       .setServiceTime(7)
37       .build();
38     Delivery n5 = Delivery.Builder.newInstance("n5")
39       .setLocation(Location.newInstance(-5, 6))
40       .addSizeDimension(CLASSIC_STUFF, 50)
41       .setTimeWindow(TimeWindow.newInstance(50, 140))
42       .setServiceTime(9)
43       .build();
44     Delivery n6 = Delivery.Builder.newInstance("n6")
45       .setLocation(Location.newInstance(7, 1))
46       .addSizeDimension(CLASSIC_STUFF, 60)
47       .setTimeWindow(TimeWindow.newInstance(10, 210))
48       .setServiceTime(10)
49       .build();
50     Delivery n7 = Delivery.Builder.newInstance("n7")
51       .setLocation(Location.newInstance(10, 8))
52       .addSizeDimension(CLASSIC_STUFF, 20)
53       .setTimeWindow(TimeWindow.newInstance(10, 740))
54       .setServiceTime(10)
55       .build();
56
57     vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n4, n5, n6, n7));
58
59     /* TASK 1 END */
60
61     IncompleteCostMatrix.Builder costMatrixBuilder = IncompleteCostMatrix.Builder.newInstance();
62
63     // --- roads
64     Set<RelationKey> roads = new HashSet<>();
65     roads.add(RelationKey.newKey(n0, n2));
66     roads.add(RelationKey.newKey(n1, n2));
67     roads.add(RelationKey.newKey(n1, n4));
68     roads.add(RelationKey.newKey(n5, n7));
69
```

Task: Q11JSALL (3/5)

Task (continuation)

```
70      for(RelationKey key : roads)
71        costMatrixBuilder.addTransportTime(key.from, key.to, roadFunction(key.from, key.to));
72
73      // --- highways
74      Set<RelationKey> highways = new HashSet<>();
75      highways.add(RelationKey.newKey(n0, n3));
76      highways.add(RelationKey.newKey(n0, n4));
77      highways.add(RelationKey.newKey(n0, n3));
78      highways.add(RelationKey.newKey(n1, n5));
79      highways.add(RelationKey.newKey(n1, n6));
80      highways.add(RelationKey.newKey(n1, n7));
81      highways.add(RelationKey.newKey(n2, n6));
82
83      for(RelationKey key : highways)
84        costMatrixBuilder.addTransportTime(key.from, key.to, highwayFunction(key.from, key.to));
85
86      IncompleteCostMatrix cm = costMatrixBuilder.completeTransportTimeMatrix().build();
87      vrpBuilder.setRoutingCost(cm);
88
89      /* TASK 2 BEGINNING */
90      // Vehicle type definition
91      VehicleType classicStuffType = VehicleTypeImpl.Builder.newInstance("classicStuffType")
92        .addCapacityDimension(CLASSIC_STUFF, 200).build();
93
94      // Vehicle instance defintion
95      VehicleImpl classicStuffInstance = VehicleImpl.Builder.newInstance("classicStuffInstance")
96        .setType(classicStuffType)
97        .setStartLocation(n0)
98        .build();
99
100     // Add vehicle instance to the problem
101     vrpBuilder.addVehicle(classicStuffInstance);
102
103     /* TASK 2 END */
104
105     VehicleRoutingProblem vrp = vrpBuilder.build();
106   }
107
108   public static double highwayFunction(Location end1, Location end2) {
109     return 1.5 * EuclideanDistanceCalculator
110       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 3;
111   }
112
113   public static double roadFunction(Location end1, Location end2) {
114     return 3 * EuclideanDistanceCalculator
115       .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 5;
116   }
117 }
```

Task 1: From the listing above, copy and paste the code betweenthe comment "TASK 1 : BEGINNING" and the comment "TASK 1 END". Complete the copied code for the network (including depot and customer definitions) so that it conforms to the target state depicted in the illustration.

Enter answer here

Task: Q11JSALL (4/5)

Task (continuation)

Task 2: From the listing above, copy and paste the code betweenthe comment "TASK 2 : BEGINNING" and the comment "TASK 2 END". Complete the copied code for the network (including depot and customer definitions) so that it conforms to the target state depicted in the illustration.

Enter answer here

Correct solution

Task 1:

```
12 Location n0 = Location.newInstance(2, 4);
13
14 Delivery n1 = Delivery.Builder.newInstance("n1")
15    .setLocation(Location.newInstance(2,0))
16    .addSizeDimension(NEW_STUFF, 50)
17    .setTimeWindow(TimeWindow.newInstance(5, 95))
18    .setServiceTime(11)
19    .build();
20
21 Delivery n2 = Delivery.Builder.newInstance("n2")
22    .setLocation(Location.newInstance(8,-8))
23    .addSizeDimension(CLASSIC_STUFF, 45)
24    .setTimeWindow(TimeWindow.newInstance(10, 99))
25    .setServiceTime(12)
26    .build();
27
28 Delivery n3 = Delivery.Builder.newInstance("n3")
29    .setLocation(Location.newInstance(-10,-6))
30    .addSizeDimension(CLASSIC_STUFF, 40)
31    .setTimeWindow(TimeWindow.newInstance(10, 99))
32    .setServiceTime(12)
33    .build();
34
35 Delivery n4 = Delivery.Builder.newInstance("n4")
36    .setLocation(Location.newInstance(-4, -2))
37    .addSizeDimension(CLASSIC_STUFF, 35)
38    .setTimeWindow(TimeWindow.newInstance(20, 290))
39    .setServiceTime(7)
40    .build();
41
42 Delivery n5 = Delivery.Builder.newInstance("n5")
43    .setLocation(Location.newInstance(-5, 6))
44    .addSizeDimension(NEW_STUFF, 50)
45    .setTimeWindow(TimeWindow.newInstance(50, 140))
46    .setServiceTime(9)
47    .build();
48
49 Delivery n6 = Delivery.Builder.newInstance("n6")
50    .setLocation(Location.newInstance(7, 1))
51    .addSizeDimension(NEW_STUFF, 60)
52    .setTimeWindow(TimeWindow.newInstance(10, 210))
53    .setServiceTime(10)
54    .build();
55
56 Location n7 = Location.newInstance(10, 8);
57
58 vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3, n4, n5, n6));
```

Task: Q11JSALL (5/5)

Correct solution (continuation)

Task 2:

```
100 // Vehicle type definition
101 VehicleType newStuffType = VehicleTypeImpl.Builder.newInstance("newStuffType")
102   .addCapacityDimension(NEW_STUFF, 160).build();
103
104 // Vehicle instance defintion
105 VehicleImpl newStuffInstance = VehicleImpl.Builder.newInstance("newStuffInstance")
106   .setType(newStuffType)
107   .setStartLocation(n7)
108   .build();
109
110 // Add vehicle instance to the problem
111 vrpBuilder.addVehicle(newStuffInstance);
```

Evaluation 2020, 2021

Task 1: 5 Points.
Task 2: 5 Points.