An Approach to Cross-Domain Situation-Based Context Management and Highly Adaptive Services in Pervasive Environments

Zakwan Jaroucheh

Submitted in partial fulfilment of

the requirements of Edinburgh Napier University

for the Degree of

Doctor of Philosophy

School of Computing

January 2012

Abstract

The concept of context-awareness is widely used in mobile and pervasive computing to reduce explicit user input and customization through the increased use of implicit input. It is considered to be the corner stone technique for developing pervasive computing applications that are flexible, adaptable, and capable of acting autonomously on behalf of the user. This requires the applications to take advantage of the context in order to infer the user's objective and relevant environmental features. However, context-awareness introduces various software engineering challenges such as the need to provide developers with middleware infrastructure to acquire the context information available in distributed domains, reasoning about contextual situations that span one or more domains, and providing tools to facilitate building context-aware adaptive services.

The separation of concerns is a promising approach in the design of such applications where the core logic is designed and implemented separately from the context handling and adaptation logics. In this respect, the aim of this dissertation is to introduce a unified approach for developing such applications and software infrastructure for efficient context management that together address these software engineering challenges and facilitate the design and implementation tasks associated with such context-aware services. The approach is based around a set of new conceptual foundations, including a context modelling technique that describes context at different levels of abstraction, domain-based context management middleware architecture, cross-domain contextual situation recognition, and a generative mechanism for context-aware service adaptation.

Prototype tool has been built as an implementation of the proposed unified approach. Case studies have been done to illustrate and evaluate the approach, in terms of its effectiveness and applicability in real-life application scenarios to provide users with personalized services.

Acknowledgments

I would like to thank my supervisors Dr. Xiaodong Liu and Mrs Sally Smith, my PhD panel chair Dr. Michael Smyth for all their help, support, expertise and understanding throughout my period of PhD study. I would also like to thank all staff in the School of Computing at Edinburgh Napier University, especially the members of the Centre for Information and Software Systems group, for providing me with valuable feedback and suggestions during my PhD study.

I am most indebted to my beloved wife, Sheren, for her endless support and understanding during the period of my PhD study. Finally, great appreciation and thanks to my mother and my parents-in-law. Although they are in Syria, they always encourage me on the phone and give me consistent spiritual support. I am proud of them and appreciate what they contribute to my life.

Publications from the PhD Work

Journal Articles

[1] Zakwan Jaroucheh, Xiaodong Liu, Sally Smith, "Recognize contextual situation in pervasive environments using process mining techniques," Journal of Ambient Intelligence and Humanized Computing, vol. 2, Dec. 2010, pp. 53-69.

[2] Zakwan Jaroucheh, Xiaodong Liu, Sally Smith, "An Approach to Domain-based Scalable Context Management Architecture in Pervasive Environments," Personal and Ubiquitous Computing, vol. 1617-4909, Jun. 2011, pp. 1-15 *(impact factor 1.554 (2009)).*

Book Chapters

[3] Zakwan Jaroucheh, Xiaodong Liu, Sally Smith, "A Software Engineering Framework for Context-aware Service-based Processes in Pervasive Environments", In IGI Global Research Handbook "Advanced approaches and tools on emerging computing systems".

Conference Papers

[4] Zakwan Jaroucheh, Xiaodong Liu, Sally Smith, "Mapping Features to Context Information: Supporting Context Variability for Context-aware Pervasive Applications," IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technologies (WI-IAT 2010), Toronto, Canada, IEEE Computer Society, 2010. *(Acceptance rate 22.7%)*

[5] Zakwan Jaroucheh, Xiaodong Liu, Sally Smith, "Apto: A MDD-based Generic Framework for Context-aware Deeply Adaptive Service-based Processes," ICWS, The IEEE 8th International Conference on Web Services, Florida, USA, IEEE Computer Society, 2010. *(Acceptance rate 17.6%)*

[6] Zakwan Jaroucheh, Xiaodong Liu, Sally Smith, "A Model-driven Approach to Flexible Multi-Level Customization of SaaS Applications," SEKE, The 22nd International Conference on Software Engineering and Knowledge Engineering, San Francisco Bay, USA, 2010. *(Acceptance rate 33.0%)*

[7] Zakwan Jaroucheh, Xiaodong Liu, Sally Smith, "CANDEL: Product Line Based Dynamic Context Management for Pervasive Applications," CISIS, pp.209-216, 2010 International Conference on Complex, Intelligent and Software Intensive Systems (ARES/CISIS 2010), Krakow, Poland, IEEE Computer Society, 2010.

Workshops

[8] Zakwan Jaroucheh, Xiaodong Liu, Sally Smith, "A Perspective on Middleware-Oriented Context-Aware Pervasive Systems," COMPSAC, vol. 2, pp.249-254, 2009 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09), IEEE Computer Society, 2009.

[9] Zakwan Jaroucheh, Xiaodong Liu, Sally Smith, Huiqun Zhao, "Lightweight Software Product Line Based Privacy Protection Scheme for Pervasive Applications, " 2011 35th IEEE International Computer Software and Applications Conference (COMPSAC'11), Munich, Germany, IEEE Computer Society, 2011.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Problem Statement

Typically, context-aware systems are composed of sensors, actuators, application components, and context processing components that manage the flow of context information between the sensors/actuators and applications. Context-awareness is considered to be the corner stone technique for achieving the pervasive computing vision. Therefore, a strong trend in context-awareness research is clearly visible in the last few years. So far, although many approaches and corresponding mechanisms have been proposed by the research community and industry, fully automated and perfectly effective context-aware services are still not a reality due to the complexity and diversity of context mining/management and the challenging nature in the consequent service adaptation. In this thesis we attempt to develop a new approach and related mechanisms to address the research question of how to achieve a perfectly effective and automated context-awareness in software services. The context mining/management is integrally linked with the consequent service adaptation in our approach.

In general, the research efforts in the context-aware service engineering domain can be roughly divided into the following categories:

(i) Context modelling and abstraction.
(ii) Context management middleware.
(iii) Contextual situation recognition.
(iv) Service design, development, and evolution techniques.

However, the research in this domain still has to address a number of challenges and problems associated with these categories:

(i) Context modelling and abstraction

Different approaches and techniques have been proposed to context modelling and reasoning. One of the most prominent technologies for this purpose is ontologies. These approaches produce ontologies that describe context information and provide means for reasoning and inference. Usually these approaches rely on Resource Description Framework (RDF) and Web Ontology Language (OWL-DL) and are combined with middleware to provide more complete context management. The main problem with these approaches is that reasoning in OWL-DL is computationally expensive. As the context manger is expected to administer a large volume of context information represented by RDF triples in the context repository, applying the reasoning capability to infer new context knowledge may have a severe impact on the overall performance of the system. Thus, limited reasoning performance reduces the applicability of these approaches in real world applications.

On the other hand, applications use context queries to retrieve the set of context information that adhere to some conditions. The application developer may not have enough knowledge about context semantics, in order to describe context queries correctly.

Finally, in order for the middleware to serve different types of applications, it should provide context-specific programming abstraction or constructs that model the *context variability*. Indeed, different context knowledge could be extracted from the context repository by focusing on different views of the context information. For example, in the smart meeting room, a seat may be equipped with light and temperature sensors to reason about its occupation. The seat could be either free or occupied. Two occupation variants may be identified: occupied by an object and occupied by a person. These variants represent two facets of the same fact. To the author’s best knowledge, the existing approaches do not provide application developers with software constructs through which a view-based customization of the context knowledge could be expressed.

As an attempt to overcome these limitations, this thesis introduces the *context variability*, *context primitive,* and *context feature* concepts. Each application expresses its interest in context information by specifying a set of context features. Each feature corresponds to a set of context primitives which will be used to generate a per-application customized contextual knowledge. Obviously, considering only the relevant context primitives would improve the reasoning performance.

(ii) Context management middleware

In pervasive environments, context management systems are expected to administer large volumes of contextual information that are captured from different areas (domains). Research in context-aware computing has produced a number of middleware systems for context management to facilitate the communications between applications and context sources. However, distributed context management among these domains raises main issues that have been neglected or partially addressed in the current approaches.

Firstly, in distributed context management scenarios, applications need to have a mechanism allowing them to identify which context management system provides the context information they are interested in. In addition, these applications need to specify domain-based context query, i.e., context information provided by context providers in specific domains.

Secondly, the existing middleware solutions have either limited generality or scalability. Some recent middleware focus on a specific application types, e.g., smart room [1] or Web content adaptation [2]. Other middleware offer distributed platforms for context management (e.g. [3][4]), federation of context management systems (e.g. [5][6]), or peer-to-peer interaction approaches (e.g. [7]). In general, the first approaches assume context model homogeneity in the distributed environment and they focus on efficient context information dissemination among distributed clients, which is only one requirement of such a distributed scenario. The second approaches provide mechanisms that allow aggregation of independent context management systems by sharing their context models with other context management systems and by providing a common interface for applications to query. Thus they support generality and interoperability among different domains. The third approaches establish a direct connection to each context management system that contains context information to be involved in evaluating application context query. Typically the distribution of context information should be transparent to the applications in the sense that they should be alleviated from the tasks associated with retrieving the context information available in different domains. In this respect, the context management system maintains all the details of how the context is retrieved, and by such enables transparent context access for providers and consumers, independently if the context information is local or not to its current domain. Although distribution is generally transparent to applications this may degrade the overall system performance as this may requires contacting several context management systems to handle application queries.

Finally, the distributed context information among different domains raises other issues such as privacy and cross-domain reasoning. That is, in order to understand the user's behaviour we may need to consider the user context information originated from the different domains the user visits. The existing middleware solutions do not provide an infrastructure that facilitates this kind of reasoning. In addition, the distribution of the user's context information among the different visited domains may weaken the privacy enforcement support.

(iii) Contextual situation recognition

Situations, the semantic interpretations of context, provide a better basis for selecting adaptive behaviours than context itself. The ability to recognize and monitor the user's situation is vital to achieve less-intrusive interaction in pervasive environments. Situation recognition is related to the activity recognition research area. For example, Loke [8] states that activity can be considered as a type of contextual information which can be used to characterize a situation. Generally speaking, activities can be seen as a sequence of events, and situations as a sequence of activities. Thus, some of the existing activity recognition could be generalized to recognize situations. This thesis focuses on abstracting from activities to infer the current situation.

To recognize activities, a reasoning process uses the sensor data to infer which activities are "occurring" at a particular point in time. This involves matching sensor data or its more abstract and meaningful form against a predefined model of activities. On the other hand, since situations are semantic abstractions from activities, human knowledge and interpretation of the world must be integrated into a model or situation representation [9]. This can be achieved in three ways: (i) a human defines the situations and their relationship based on his knowledge during a specification process using rule-based or ontological approaches (e.g. [10][11][12]), (ii) situation models are learned from training data by associating a human-defined situation label via learning techniques (e.g. [13][14][15]), (iii) or the situation model is derived from a combination of both (e.g. [16]).

Learning approaches have been widely used for activity recognition, due to their ability to automate the creation of the activity model from training data and to handle noisy sensor data. On the downside, training data can be difficult and costly to acquire. On the other hand, when contextual situations and application needs of situations are known in advance, a human expert can specify the situations manually. However, expert hours are expensive; in addition, this model is not grounded in physical observations. Therefore, a trade-off solution could be to create an initial situation model with minimum expert knowledge input, and then her knowledge is exploited to provide situation labels (such as in [16]).

Defining the user’s situation may require considering the different states (activities) the user experiences in the different spatial domains they visit. For example, to identify if the current day was busy for the user there is a need to consider the different activities and states the user has experienced in work, shopping, on the road, etc.

At present, there is no generic solution capable of recognizing situations from perceptual events coming from different spatial domains and explicitly support knowledge about activities the user experiences in the visited domains. This generic solution should regroup different layers of abstractions where the multi-domain activity sequences in one layer are fused in the situation recognition layer to recognize situations spanning one or more domains.

In addition, typically the existing approaches require constructing sequence-based activity models comprising low-level activity features and trying to recognize activities that follow this model. However, to recognize situations, this thesis argues that in reality human behaviour may not follow a specific sequence of activities; rather situations may have distinct series of activities but with no particular sequence. Thus, relying on sequences of activities may limit the accuracy of situation recognition (e.g. [17]). Finally, a generic solution should provide the flexibility to define situation models based on the domain expert knowledge or on training data. This way, the situation designer has the option to define the situation model from scratch or to use the sensor data to infer (mine) the user behaviour and then manually identify the situations of interest.

(iv) Service design, development and evolution techniques

As Web service architecture is a popular trend in the service domain, it is not surprising that most of the existing approaches follow Web service technologies. On the other hand, the existing approaches tackle the context-aware adaptation either on the client side or server side. This thesis focuses on the server-side solutions.

In general the context-aware service engineering can be regrouped in three categories [18]: source-code level approaches, message interception approaches, and model-driven approaches. In the source-code level approaches, the business logic of the service can be enhanced with code fragments performing context handling and the required adaptive behaviour. This is achieved either by extending the programming language syntax or providing external context handling mechanisms [18]. However, these approaches may be not suitable for the context-aware service development which usually involves several stages (e.g. analysis and design) prior to the actual code development. In the message interception approaches, the context handling and adaptation is performed by intercepting and modifying the incoming and outgoing messages of a service without affecting the core business logic of the service. However, this may be insufficient as the change in user context or business rules may require changing service business logic as well. Relying heavily on models, the model driven approaches are promising approaches as they consider the context-aware adaptation in the full software development life cycle. In addition, they enjoy the inherited power of model transformations and (semi)automatically production of executable code. Thus, this thesis focuses on proposing model-driven based approach for service adaptation.

Variability refers to a system's ability to be changed, extended, customised or configured for use in a particular context [19]. Usually developing a context-aware adaptive service requires the developer to specify kinds of variations (i.e. variation points and variants) in the service model that will be determined at design time or runtime according to the operating context. However, this may pose three main problems from the developer point of view: (i) the variation points and variants are sometimes embedded in the service logic itself (e.g. VxBPEL [20]) which weakens the system modularity and violates the separation of concern principle, (ii) the constructs used to specify the service variant (i.e. variation points and variants) do not reflect the way the developer or designer logically view the difference in the service model in each context usage, and (iii) managing the variation points and their dependencies becomes a difficult task when the number of these variation points increases.

1.2 Aim and Objectives of the Research

Motivated by the problems and directives in mind mentioned in Section 1.1, this thesis proposes a unified approach for developing context-aware services and contributes to the knowledge by addressing the following points:

1- In order to support the developer in defining the context queries, a hybrid approach to context modelling that combines the ontologies with a feature-oriented modelling technique is needed. That is, the available context information is “promoted” using context features that could be shared among different applications. Each context feature corresponds to a specific set of context primitives as will be seen. Obviously, considering only the relevant context primitives would improve the reasoning performance and reduce response time which is a vital issue in the pervasive environment. In addition, the reusability principle is respected and the developers are able, by configuring the context feature model, to get the context information they are interested in. To this end, ideas from Software Product Line techniques (feature model) have been leveraged.

2- The complexity of developing context-aware applications makes the existence of middleware a vital requirement. Thus this thesis focuses on developing and validating a distributed domain-based context management middleware. It proposes *ubique*, a new middleware architecture which is adequate for addressing the context consumer requirements anytime and at any place in future pervasive environments. *ubique* allows applications to describe and maintain context queries that involve context provided by various environments (domains). By incorporating the management and communication benefits of the Jabber protocol and taking advantage of the semantic and inference benefits of ontology-based context models, *ubique* forms an underlying robust and generic infrastructure for cross-domain context dissemination, which significantly simplifies the development of context-aware pervasive applications.

3- This thesis aims also to take advantage of the distributed context management architecture, *ubique*, to capture and reason about the different contextual situations which span one or more domains. To recognize such situations, this thesis focuses on the potential use of Process Mining techniques for firstly mining the actual behaviour and secondly comparing the real situation of the user with the expected situation.

4- Services in pervasive environments need to cope with high variability, as they are deployed on a diversity of computing platforms, operate in different execution environments, and provide personalized services according to dynamically varying users' requirements. Supporting the development, provision and evolution of such services in this setting requires a unified solution that integrates the cutting edge technologies from several related areas, such as context-awareness and adaptiveness, into a seamless consistent approach. Therefore, this thesis also proposes an automated model-driven approach for the development and evolution of highly agile context-aware services.

1.3 Contributions to Knowledge

In order to address the research question of how to achieve an effective and automated context-awareness in software services, this thesis proposes a new approach to facilitate the developer task of designing and implementing context-aware adaptive services. It attempts to solve some of the problems associated with the context modelling and management, cross-domain contextual situations recognitions, and the context-aware service adaptation. Thus, this dissertation provides four main contributions:

- **Product line based context information representation.** This representation significantly enhances reusability of context information by providing context features to satisfy different application needs. This allows the context modeller to specify the context information in a high-level and logical way that regroups context variabilities; and provides application developers with context-specific programming constructs to express their needs. The result is a more intuitive way to represent context and improvement of overall systems performance.
- **Process mining based situation recognition.** Since the aim of pervasive computing and ambient intelligence is to enable users to interact with the environment in an intelligent way, the applications should not only consider the current relevant context information but also their history and their distribution among different domains. Thus the second contribution is the introduction of a formalism for the situation recognition problem and the leverage of process mining techniques for measuring situation alignment, i.e., comparing the real situations of users with the expected situations which span one or more domains.
- **Jabber-based cross-domain context management middleware.** The third contribution is the leverage of Jabber protocol to create mechanisms that address the requirements of scalable distributed context management, privacy enforcement, and efficient context information dissemination and query handling. In this respect, *ubique*, middleware architecture is proposed. *ubique* incorporates the management and communication benefits of Jabber, while also taking advantage of the semantic and inference benefits of ontology-based

context models. This architecture establishes a robust cross-domain context management and collaboration framework which has been designed, implemented and evaluated.

- **Model-driven mechanism for context-aware adaptive services.** The forth contribution consists of proposing a generative approach for the development and evolution of services. This approach supports the viewpoint of context-aware adaptation as a crosscutting concern with respect to the core "business logic" of the service. In this way, the design of the service core can be decoupled from the design of the adaptation logic. This means that the task of core service design can be separated from the adaptation logic design task and they could be considered as two separate concerns. To this end, ideas from the domain of model-driven development (MDD) and generative programming have been leveraged.

Based on the successful application of existing technologies, such as MDD, Jabber protocol, generative programming, and software product line, the proposed approach contributes to (i) provide a new context modelling approach that facilitates the developer task, (ii) to allow the developers to design and recognize cross-domain contextual situations, (ii) a new domain-based context management infrastructure that allow the developer to define domain-based queries and ensure user's privacy, and (iv) to provide the developer with tools methodology that captures the service variants in a logical way to design and implement context-aware adaptive services. Thus, the contribution aims towards a software engineering approach which takes into consideration the ease of developing context-aware services.

1.4 Statement of Methodology

The research work in this thesis has been accomplished with a methodology combining literature review, creative research on approach process and novel technologies, manual and tool-based case studies to verify, evaluate, and then refine the approach and tool set.

Firstly, a comprehensive literature review has been done to get a complete view of the current state of art of the related areas. Based on this view, crucial problems of designing and implementing context-aware adaptive service and context modelling have been identified. A proposed framework of the approach has been established to correct the identified problems.

In next stage, case studies have been conducted, both manually and tool-based on the adaptation of context-aware services. The result has triggered further improvements and refinements to the developed approach. The related prototype tool set has been designed and implemented on top of the approach. Tool-based case studies have been done to evaluate the tool set and the approach. The experiment data has been used to improve both the tool set and the approach.

Papers have been submitted to top international conferences and journals to disseminate the research results and to get valuable feedbacks.

1.5 Criteria of Success

In order to address the research question, a set of criteria have been identified to measure the success of the proposed approach. For example, concerning the context modelling approach, is it possible for the modelling approach to take advantage of the ontology-based context modelling approach and the same time improve the system performance by reducing the reasoning time? Can the resulting context model be reused and shared by different types of applications? Can the respective modelling approach provide a mechanism through which the context variability can be expressed in order to provide the applications with the context information they need on different levels of abstractions?

Further, since the context information is naturally distributed, the context management should be distributed in order to allow efficient and scalable dissemination of context. In this case, additional restrictions may arise when the mobile users roam across domains (e.g. concerning limited connectivity and bandwidth, unknown network conditions, etc.). Accordingly, is it possible to provide the developers with mechanisms to define their queries about context information of interest which may span different domains? Is it possible to design an efficient protocol for exchanging context information between domains that disseminates only the required information and at the same time provides a mechanism to enforce the user's privacy? Can the respective context management middleware provide an infrastructure that permits to recognize contextual situations which span one or more domains?

On the other hand, services evolve according not only to the available context information but also to the changes in the business rules and requirements. Can the service core logic be designed and implemented separately from the context handling and adaptation logics through the service development and evolution phases? Is it possible to provide the developers with mechanisms to capture the service variability from a logical point of view in order to easily manage and understand the service variants in each usage context?

The aforementioned criteria are described and discussed in more details in Chapter 3.

1.6 The Structure of the Thesis

The thesis is organized as follows:

Chapter 1 gives an introduction of the research, including the problem statement, the aim and objectives of the research, and the contributions to knowledge.

The literature review is presented in Chapter 2, which includes the current state of context modelling and management, context-aware service adaptation, and the current state of enabling technologies such as software product lines, model driven architecture, Jabber protocol, and process mining.

Chapter 3 summarises the related research projects in context modelling and management, situation recognition, and service-based application adaptation, and describes typical research projects in detail. In addition, these projects are critically analysed and a conclusion is drawn, which gives the motivation of the research.

Chapter 4 provides an overview of the proposed approach and shows how the different parts of the approach are interlinked.

Chapter 5 presents the context modelling approach, which includes the product line based context model, the mapping between context features in the context feature model and the available context information, and the algorithm used to generate a customized view of the available context information.

Chapter 6 describes the *ubique* approach for a collaborative context management among different context servers distributed in different domains in the pervasive environment. It describes a new protocol (built upon Jabber protocol) which has been designed and implemented in order to efficiently disseminate context information between different domains in a way that respect the users' privacy.

Chapter 7 proposes a multi-layered conceptual architecture for contextual situation recognition. It formally defines the situation recognition problem and proposed a recognition approach by leveraging ideas from process mining techniques.

Chapter 8 describes *Apto*, the proposed approach for service development and evolution. It provides a conceptual model for the context-aware adaptive services and describes the *Apto* prototype tool, including its architecture and implementation.

Chapter 9 presents the conclusions of the research and future work.

Chapter 2 Literature Review

This chapter conducts a broad overview on context-awareness issues (such as context modelling, abstractions and management) and on service adaptations which are usually triggered or driven by context change. This chapter also conducts a survey of several techniques that have been found useful for the proposed approach such as software product lines, model driven development, process mining, and the Jabber protocol. These techniques are the foundation of the development of the proposed approach.

2.1 Current State of Context Modelling and Management

2.1.1 Introduction

With the growth of mobile devices such as laptops and smart phones, it is not surprising that mobile computing has attracted considerable attention in recent years. In an attempt to go beyond the traditional view of explicitly used computers and terminal devices, a new more general paradigm of user-centric mobility was introduced by Mark Weiser [21] in 1991 and called “Ubiquitous Computing”. In this paradigm, smart and autonomous computing technology will be embedded in every device to enhance the use of computers by making computers effectively available throughout the physical environment and, at the same time, making them invisible to the user. Mark Weiser [21] expressed this goal as achieving the most efficient technology and making computing as ordinary as electricity. Thus, instead of relying on specialized devices carried and maintained by the user such as a mobile phone, the focus is now on provisioning services to the user.

Furthermore, pervasive computing is another term used in the same context but from different points of view [22]. Pervasive computing emphasizes mobile data access, smart spaces and context awareness. Thus pervasive computing focuses on three main areas: (i) how do people see and use mobile and wireless computing devices; (ii) how to create and deploy applications to end users; and (iii) the way ubiquitous services enhance the environment. Because of this conceptual overlap, this dissertation uses the words “pervasive” and “ubiquitous” interchangeably.

2.1.2 Defining Context-awareness

The traditional model of software systems relies on the input explicitly taken from the user to act upon and produce explicit output. In pervasive environment, this model is being seen as unsuitable, where users find themselves dealing with a large number of services and devices. Thus, services have to operate not only on the explicit input but also on implicit information gathered from the environment; in other words, they have to be context-aware so that they can adapt to changes in situations on behalf of the user.

In the literature, there are many different definitions and uses of the term context (e.g., [23][24][25][26]). Definitions given by earlier works agree on the key idea that contexts describe situations. For example Dey [23] confirmed this by defining context as: “Any information that can be used to characterize the situation of an entity. An entity is a person, a place, or a physical or computational object that is considered relevant to the interaction between a user and an application, including the user and application themselves.”

Context can also be defined as meta-information to characterize the specific situation of an entity and to describe a group of conceptual entities [24]. Winograd [25] indicated that in using open-ended phrases such as "any information" and "characterize", the context becomes so broad that it covers everything. He indicated also that “something is context because of the way it is used in interpretation, not due to its inherent properties. The voltage on the power lines is considered as a context if the interpretation of the user’s or computer’s action is dependent on it, but otherwise it is just part of the environment. Therefore, “context depends on the interpretation of the operations involved on an entity at a particular time and space rather than the inherent characteristics of the entity itself." Another interesting definition for context [26] indicates that "context is always related to a focus". Viera et al. argue that context should always be considered related to a focus, which is a step in a task execution, in a problem solving or in a decision making process. Moreover, the context evolves dynamically according to the focus, which enables a context-aware system to separate relevant from not relevant knowledge in order to determine the context.

Context-awareness is considered as an important functionality in pervasive computing. For example, a context-aware mobile phone could be switched into silent mode once the user enters a conference room. Furthermore, as stressed by the ubiquitous vision, distributed systems need not only adapt to the change in the available resources, but also to the users' preferences and profiles over time and the physical environment. This ability is generally referred to as context-awareness. Thus, context-awareness is the ability for a software system to acquire, manage, interpret, and respond to context to provide appropriate services to the changing situation [27]. Context-awareness could also be defined as the "capability of a context-aware system or middleware to provide anytime access to heterogeneous, distributed, and unanticipated context information in global scale and for distinct scenarios" [28].

Obviously, context-awareness is central to ubiquitous computing that aims at delivering applications to end-users in an opportunistic way, with the best quality possible. We can say that a system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task at hand.

2.1.3 General Concepts

In context-aware applications, any interaction is based on two elementary concepts: *entity* and *context information*. An *entity* is any object that can be represented in the computational environment, such as a user, a physical object or any computational resource. *Context information* is an abstract information that describes the entity's state and its relations with other entities [29]. For example, the entity *user* is characterized by his location and his relationships with others.

Context information representation is implemented through *context types* and their *instances*. *Context type* is a computational representation of context information which specifies its data structure. For example, the location information provided by a GPS sensor could be represented by a *GPSLocation* type which regroups three floating point numbers: *latitude*, *longitude*, and *elevation*. Context type does not only specify the state of an entity but also its relations with other entities. For example, the user's context information could be represented by a *User* type which regroups his location, current activity, role, etc., as well as his relations with other entities. Different context types could be adopted to represent abstract context information. For instance, the location information could be represented as a symbolic location (e.g. Room1, Floor2, BuildingC, etc.), or proximity-based location (e.g. [30]). Thus, each representation could be modelled by a particular context type. Different applications are prepared to deal with different context types. For example, the location context information provided as geo coordinates can be useful for applications displaying people's locations on a map but not useful for other types of applications.

Context instance is the set of values that describes the states and relations of an entity at a specific point of time and which conform to a certain context type. For example, an Alice context instance could be described as in the Figure 2.1.

Alice : Person

Latitude = “55.923215”
Longitude = “-3.286835”
IsInvolvedIn = “Reading”
sitsBesides = Bob

Figure 2.1 Example of context instance

Context model determines the set of available context types. It defines relevant concepts to the application domain which the middleware is prepared to deal with. For example, the CoBrA [31] middleware models entities such as Agent, Person, Meeting, and Schedule which supports implementing smart meeting applications. The expressiveness and complexity of a context model depends on the modelling approach adopted in the system, which defines how the concepts and their relationships are described.

The question now is how can the context information acquired from different sources (e.g. user, device and environment contexts) be formally represented, managed and integrated to be used by the application layer for adaptation. In [27] a set of necessary functional elements that context-aware systems have to support have been identified:

- Context acquisition which concerns mechanisms to obtain the context information from different context sources. Reusable context acquisition requires that the high-level context usage be decoupled from the low-level context sensing.

- Context modelling which forms the basis for context sharing and interpretation. Existing approaches to context modelling differ in their power of expressiveness, in the support they can provide for reasoning about context information, in the computational performance of the reasoning, and in the scalability of the context information management [9]. In previous works, both informal and formal context models have been proposed. Informal context models do not ease shared understanding about context as they rely on proprietary representation schemes. Among systems with informal context models is Context Toolkit [32] which represents context in the form of attribute-value tuples. Today, with the advance of context aware computing, there is an increasing need for developing formal context models to facilitate context representation, context sharing and semantic interoperability of heterogeneous systems [33].

- **Context aggregation**: Based on a shared context model, context aggregation merges interrelated information gathered from different sources and enables further data interpretation. This alleviates context-aware applications from the overhead caused by querying from distributed context sources.

- **Context interpretation**: The low-level information needs to be interpreted to derive the high-level context used by applications. Furthermore, a specific context can be translated into logical situations [34]. For example, we need to derive high-level location (e.g., which location is the user in? living room, conference hall, etc.) from related low-level information (e.g., GPS coordinate, sensors data, etc.). Currently, context interpretation (reasoning) could be achieved by several approaches such as an ad-hoc manner, rule-based reasoning (e.g., [35][36]) and machine learning.

- **Context query**: Context-aware applications need a mechanism –context query- to access interrelated information spread across distributed context repositories. Currently there exist several query languages (e.g., SPARQL language which could be used to query context information represented by RDF tuples).

The following section focuses on the different approaches for context modelling. An efficient context modelling technique should exhibit characteristics like flexibility, extensibility, expressiveness, and reasoning which are vital to enable context awareness.

2.1.4 Classification of Context Modelling Approaches

A context model is needed to define and store context data in a machine processable form. Developing context-aware applications should be supported by adequate context modelling and reasoning techniques [9]. A well designed model is quite important in any context-aware system for the provision, storage, and retrieval of context data.

Currently there are several means for context modelling; they can be regrouped in the following categories:

1- Early approaches: Key-value and Mark-up Models

Key-value models use simple key-value pairs to define the list of attributes and their values describing context information used by context-aware applications. Schilit et al. [37] used key-value pairs to model the context by providing the value of context information (e.g. location information) to an application as an environment variable. These models are easy to manage, but are not adequate for sophisticated structuring and reasoning purposes.

Mark-up scheme models integrate the model schema and values using mark-up languages such as XML. The W3C standard for description of mobile devices, Composite Capabilities/Preference Profile (CC/PP) [38], is the first context modelling approach to use a Resource Description Framework (RDF) and to include elementary constraints and relationships between context types. CC/PP is intended to express both device capabilities and user preferences. CC/PP can be considered a representative both of the class of key-value models and of mark-up models, since it is based on RDF syntax to store key-value pairs under appropriate tags. Some approaches (e.g. [39][40]) are defined as extensions to the CC/PP [38] and User Agent Profile (UAProf) [41] standards, which have the expressiveness reachable by RDF/S and a XML serialization. These kinds of context modelling approaches usually extend and complete the basic CC/PP and UAProf vocabulary and procedures to try to cover the higher dynamics and complexity of contextual information compared to static profiles.

The main critic of these approaches is their limited capabilities in: (i) capturing relationships and dependencies of context information, (ii) allowing consistency checking, and (iii) supporting reasoning on context and on higher context abstractions.

2- Graphical models have been derived from generic modelling methods such as Unified Modelling Language (UML) (e.g. [42]) and Entity Relationship Diagrams (ERD). The main critic of these approaches is that they are not well suited to capturing special features of context information such as: (i) historical information, (ii) uncertain and incomplete information, and (iii) dependencies between different types of information [43].

A more recent and interesting proposal of a graphical oriented approach to model contextual interrelationships is the Context Modelling Language (CML) (used e.g. in [44]). CML is a tool to assist designers with the task of describing types of information (in terms of fact types), their classifications (sensed, static, profiled or derived), relevant quality metadata, and dependencies between different types of information in order to specify the context requirements of a context-aware application at design time. Later, the modelling concepts of CML have been reformulated as extensions to Object-Role Modelling (ORM) [45]. However, CML has two main limitations: (i) all context types are uniformly represented as atomic facts; thus, it is not suitable for representing a hierarchical structure of context information, and (ii) as it is domain and application specific, it does not support interoperability found, for example, in ontology-based models.

3- Object-oriented models exploit the encapsulation and reusability present in an object-oriented approach. The details of context processing are encapsulated at the object level and access to context information is only through specified interfaces. Representatives of this kind of approaches include *cues* used in TEA project [46] and the *Active Object Model* of the GUIDE project [47]. The main drawback of these approaches is the lack of supporting reasoning on context information.

4- Logic-based models formulate the context as a set of facts, expressions and rules. In logic-based context models a context is (generally) defined using facts (context properties) with expressions and rules to describe and define relationships and constraints. Contextual information is added, updated or deleted from a logic-based system in terms of facts or is inferred using rules that describe and define relationships and constraints in logic-based systems. A characteristic of logic-based systems is a high-degree of formalism. An early representative of this kind of approach is the Extended Situation Theory [48] and the Sensed Context Model proposed in [49].

5- Domain-specific modelling. In the literature, there exist some works on modelling the context information that can enhance the functionalities of domain-specific context-aware applications. Two examples could be identified in this category: (i) the W4 context model [50] that supports the representation of context as (Who, What, Where, When) Linda-like tuples and provides an interface to store and query such tuples, and (ii) spatial modelling approaches that give space and location special handling.

Location is considered one of the most important pieces of context information: e.g. Schilit et al. [37] define three important aspects of context as “Where you are, who you are with and what resources are nearby”. Most spatial context models are fact-based models that organise their context information by physical location. One of the most representative examples of these approaches is the global context model called the Augmented World Model (AWM) [51] provided by the Nexus project. The Nexus project aims at providing shared context models in an open, federated environment [52]. In this respect, autonomous data servers and sensors offer different local context models, which are federated into an integrated view over those context models for the applications (the AWM). AWM is an object-oriented information model for applications that use spatial data or services that are linked to locations. Most object classes of this model inherit from the class *SpatialObject*, which makes the Augmented World model inherently spatial.

Obviously, spatial context models are well suited for context-aware applications that are mainly location-based e.g. many mobile information systems. Spatial context models allow reasoning about the location and the spatial relationships of objects such as the inclusion in some area or range and the distance to other entities.

In fact, both middleware and context models are strongly interdependent since the complexity of a context model determines the complexity of context management by middleware [53]. Therefore, since many context-aware applications use space as a primary context, it is reasonable to design context management systems to efficiently support spatial queries, e.g., by managing spatial indexes.

These domain specific approaches are important to support context-aware application in particular domains. However, an application- and domain-agnostic context model, that captures various types of context information and the dependencies between them, that could be reused and shared by different applications, is also needed in pervasive environments.

6- Ontology-based models. Currently, with the emerging Semantic Web concept, a number of open standards for exchanging machine-understandable information have been established. For example, Web ontology languages (i.e., DAML+OIL, OWL [54], and its sublanguage OWL-DL which can be viewed as expressive Description Logics, with an ontology being equivalent to a Description Logic knowledge base.) provides formal logic model to support the formal definition and sharing of domain vocabularies for resources. Therefore, the ontology-based models provide a uniform way of specifying a model's core concepts as well as an arbitrary amount of sub-concepts and facts, which facilitates sharing and reuse of contextual knowledge.

Ontology-based context models exploit the representation and reasoning power of the description logic of OWL-DL in three points: (i) the expressiveness of the language is leveraged to represent complex context information that cannot be represented by simple languages (e.g., CC/PP [38]); (ii) since ontologies provides a formal specification of the semantics of context information, it is well suited for sharing and/or integrating context among different sources and applications; and (iii) the correspondent available reasoning tools can be used both to (a) detect possible inconsistencies in the context data, and, (b) to support the reasoning task i.e. to derive new knowledge based on the defined classes and properties, and on the individual objects retrieved from sensors and other context sources.

In order to overcome the limitation of OWL-DL expressiveness, the possibility of augmenting the expressivity of ontological languages through an extension with rules has been recently investigated by the Semantic Web community and thus the SWRL language [55] has been proposed. A further research issue considers extending existing ontological languages to support fuzziness and uncertainty while retaining decidability (e.g. [56]). However, the main problem in adopting ontology-based approaches in a pervasive environment is not related to their expressiveness but to their applicability; the reasoning in OWL-DL is computationally expensive which leads to serious performance issues especially when the ontology is populated by a large number of individuals (see e.g. [33]).

Strang and Linnhoff-Popien [57] present a survey of context models. The survey evaluates different context models with respect to specific criteria including distributed composition, partial validation, quality of information, incomplete information, and level of formality. The authors conclude that object-oriented and ontology-based models best meet the criteria and that ontology-based models are the most promising for context modelling with respect to handling context in a distributed fashion, validating context, providing quality of context indicators, supporting incompleteness and ambiguity of context, and providing a formal definition of the domain. This dissertation focuses on the ontology-based approaches and tries to overcome some of the main limitations of the existing approaches which harness ontology for context encapsulation.

2.1.5 Context Information Abstractions

The limitation of low-level context when modelling human interactions and behaviour may reduce the usefulness of context-aware applications. As aforementioned, one possible solution to alleviate this problem is the derivation of higher-level context information from raw sensor values, called context reasoning and interpretation. This can be achieved by creating a new model layer that gets the sensor readings as input and generates or triggers system actions. The most common notion that has been employed to refer to this high-level context layer is the situation (see for example [12][48][58][59]). Situations permit defining high-level specification of human behaviour or other context information which helps to inject meaning into applications.

Additionally, situations are more stable and easier to define and manage than basic context information. That is, situations can be specified in different ways based on context information. For example, a user_is_busy_now situation can be specified by: (i) the user calendar and his position, (ii) his current activity and environmental noise and sound, or (iii) his to do list and time being, etc. In each case, even if the context information defining the situation changes, the situation itself remains stable and the therefore, the applications themselves remain stable as the system actions are associated to this situation. Adaptations in applications are then triggered by a change of the situations (caused by context information change). This leads us to the situation-awareness concept which refers to the capability of the entities in pervasive computing environments to be aware of situation changes and automatically adapt themselves to such changes to satisfy user requirements, including security and privacy [12].

The question now is how to define and represent situations. As situations are human perceptions of low-level context information, human knowledge and interpretation of the world should obviously be embedded when defining situations. Thus, we have two options: (i) either the human manually defines the different situations based on their knowledge, or (ii) by using machine learning the situations are automatically recognized and learned (e.g. [60][16]). In the latter learning-based approaches, the recognition rate depends on the number and kind of observations provided for recognition and situations to be recognized. For example it is 88.8% in McCowan et al.'s work [60] on recognizing the group actions in meetings based on the interactions of the individual participants, and 94.3% in [16] which learns situation models by supervised learning algorithm using feedback from users.

However, these approaches require a training phase during which an important number of situation examples (which may require significant period of time) are collected and analyzed and which require human intervention (e.g. for situation labelling). For example, if an application needs to recognize a pick-pocket situation in a shopping mall, we would need at least one or more pick-pocket scenarios which may only take place once a month.

Most of the existing approaches refer to Dey's definition of situation: "description of the states of relevant entities". Thus, a situation is a temporal state within the context. The approaches in this category usually use formal logics to represent these states (e.g., [33][48][58][10]). For example, in [33], logic reasoning has been used to reason over low-level, explicit context to derive high-level, implicit context i.e. situation. Thus, these approaches provide high-level of abstraction and formality for specifying situations. However, as the context information is incomplete and ambiguous in pervasive environments, these approaches may not be well suited to recognize situations. To cope with this, some approaches (e.g. [61]) try to combine first order probabilistic logic (FOPL) and web ontology language (OWL) ontologies, to provide a common understanding of contextual information to facilitate context modelling and reasoning about imperfect and ambiguous contextual information.

Recognizing situations in the context information could be computationally expensive. Thus, to reduce the search space for potential situations, some approaches (e.g. [62][63]) focus on defining the relationships between situations (e.g. represented by Allen's temporal logic [64]). This way, by knowing the current situation, the search for situations could be limited to those situations having potential occurrence (e.g. successor of the current situation). In addition, these approaches model the behaviour within the environment which can be described by a sequence of situations and their relationships. These approaches suffers a limitation from the applicability point of view; i.e. as at least one situation is active at one time, this requires that all potential situations, their relationships and transitions are included in the model which is a difficult task and may not be always possible.

2.1.6 Context Management Middleware

In pervasive environments, the context-aware application adaptations are usually driven by the change in context information. This information can be originated from different sources (e.g. sensors). For example, in the smart meeting rooms, the presenter location may be provided by a proximity sensor to identify if the presenter is inside the room or by using a microphone connected to voice recognition software to identify certain people inside the room. Moreover, this context information may be used by different applications. For example, both the presenter location and the current situation in the meeting room can trigger the presentation transfer application so that the presentation will be projected to the closest screen. In addition, this information may be also used by a cameraman system that automatically records the presentation and selects, based on the current presenter location and situation, the appropriate camera to record the presenter, the audience, etc. This requirement of reuse calls for middleware systems that alleviate developers from developing context-aware applications from scratch.

Middleware for context-aware systems refers to the components located between the application layer and the sensors layer in addition to the communication framework connecting the distributed components together. Therefore, the main goal of the middleware in context-aware computing is to decouple the communication between context providers (e.g. sensors), and the applications interested in this information.

In addition, development of ubiquitous application is a complex and error-prone task because they must cope with heterogeneous infrastructures and with system dynamics in an open network. The role of the middleware is therefore essential to support mobility and adaptation of applications to the current context [65]. Thanks to the abstraction provided by the middleware, it is able to hide the heterogeneity of the networking environment, support advanced coordination models among distributed entities and make the distribution of computation as transparent as possible [66].

Typically these middleware systems adopt an asynchronous communication mechanism such as publish/subscribe [67] or tuple-space [68], as a basis of interaction between context providers and applications. This mechanism allows applications to specify the context information they need (i.e. their interests) and to asynchronously receive notification events that match these interests.

The context manager (CM) is an architectural middleware component responsible for storing context information, managing applications subscription to context changes, and handling the registered applications' queries. It is an independent infrastructure that enables interaction between context provider and consumer. CM is also responsible for managing the context model and validating the consistency of the context instances according to the model. Of course, the underlying context modelling approach plays an important role in defining the complexity of implementing the CM and its performance. For example, ontology-based models require constant execution of inference rules which usually degrade the CM performance.

The context information is naturally distributed in different spatial domains in the pervasive environment. In [29], the pervasive environment is organized hierarchically by dividing it into context domains and sub-domains. A context domain is defined as an abstraction of a spatial area which has a clear boundary and it is built on top of the traditional notion of network domain. The context domain establishes the CM scope. A CM should allow the automatic discovery, retrieval and exchange of the context information distributed in different domains.

Although users and applications are more interested in context information available in their local domain, other context information from other domains may also be relevant to the current task at hand. Thus, a collaborative context management across domains is needed.

2.2 Current State of Service Adaptation

2.2.1 Introduction

This thesis focuses on context-aware applications developed using the Service Oriented Architecture (SOA) guidelines. SOA is a promising way to address the problems of the integration of heterogeneous applications in a distributed environment [69]. In a SOA environment, every service provider has to declaratively define the functional and non-functional requirements and capabilities of their services in an agreed machine-readable format. In its basic form, SOA model requires that service provider publish its services' descriptions in a public registry; service requestors discover services by querying this registry; the service requestors then select and bind to the selected services dynamically.

Three components can be identified at each end of interaction between services: (i) the service implementation or business logic, (ii) the service metadata describing the requirement and capabilities of the service that can be used by other parties to understand the service functionality and how to interact with it, and finally (iii) the SOA middleware which supports the automatic service discovery, selection and dynamic binding. The decoupling of implementation achieved by separating and publishing service interface definition is generalized in SOA environment to include not only the functional aspect of the service, but also the quality of service and middleware interoperability aspects [69].

The Web services framework is an instance of an SOA. Web services are a well-known XML-based application-to-application communication technology that is built upon standard internet protocols such as SOAP, WSDL, UDDI and XML. The basic component of Web services is the Simple Object Access Protocol (SOAP), a XML based communication protocol for interacting with Web services. The services are described using Web

Services Description Language (WDSL). It describes the service location, the supported operations and the format of messages to be exchanged between service providers and service requestors. Universal Description Discovery and Integration (UDDI) allows service providers to advertise their services in a standard way and for service requestors to query services of their interest.

From an architecture point of view, SOA may represent an effective architectural paradigm for the design of pervasive applications [70]. That is, the loose coupling and interoperability properties may provide a good support for the realization of flexible applications that can be easily adapted to different execution contexts.

By introducing a layer of abstraction above the operational systems layer, Web services eases interoperability between heterogeneous systems running on different platforms, managed by different providers, and implemented in different programming languages. The power of Web services is the ability to combine Web services possibly from different providers in order to create value-added and feature rich integrated services. For example, a hotel service, an airline booking service and a credit card service can be composed into a travel booking service.

Web service has to cope with the highly dynamic pervasive environment. Web services standards are inadequate on rendering Web services adaptable and aware of the changes in Web service capability or availability as well as user's context. This is due to the request-response pattern imposed by interacting Web services with other peers and users [71]. This means that the Web service replies to requests without assessing (i) its execution capabilities and internal execution status, and (ii) surrounding environment as well as the information describing users and their preferences prior to binding to any composition. In other words, Web service should be context-aware [24]. In this respect, three main overlapped research areas could be identified: context-aware service discovery, context-aware service composition, and context-aware service adaptation.

2.2.2 Context-aware Service Discovery

To operate in dynamic and potentially unknown environments a mobile client must first discover the local services that match its needs, and then interact with these services to obtain the required application functionality. It has been shown that incorporating context and situation awareness in service discovery can greatly improve the precision and recall of the discovery results [72], where recall is defined as number of relevant services retrieved in service discovery divided by the total number of relevant services available; and precision is defined as the number of relevant services retrieved in service discovery divide by the total number of services discovered.

Service discovery approaches basing on the comparison made at the syntactical level (i.e., compare inputs, outputs, pre-conditions and post-conditions to match the appropriate component services) may raise semantic incompatibility. As the user of a service, being the user a human or an application, is interested in a given functionality provided by this service and not on how this functionality is implemented, an abstraction of services is needed (e.g. [73]).

2.2.3 Context-aware Service Composition

Service composition refers to the technique of creating composite services with the help of smaller, simpler and easily executable services or components [74]. Web services from different sources and locations can be identified, selected and composed to achieve a certain task. Composing services rather than accessing a single service offers greater benefits to users. Thus, composition addresses the situation of a client request that cannot be satisfied by any available service, and combining a set of services into a composite service might be used for fulfilling the request [75].

A composite service (also known as a process) is always associated with a specification which describes the list of component Web services that participate in the composition, their execution chronology and types of dependencies between them. Examples of Web composition languages are Business Process Execution Language (BPEL), and Web Services Flow Language (WSFL). The main objective of these languages is to provide high-level description of the composition process. Currently, WS-BPEL is considered the de-facto industry standard for orchestrating Web services. It is used to model the behaviour of processes with XML-based script.

Research in service composition has followed two directions: one direction defines languages to formally describe services and composite services in terms of e.g. service input/output, service constraints and invocation mechanisms. This research also includes developing engines that utilize these languages to generate workflow specifications that compose different services. The other direction concerns development architectures that enable service composition. Based on a declarative description of services, these architectures perform the task of discovering, integration and execution of the relevant services.

2.2.4 Context-aware Service Adaptation

A software application is adaptable if it can change its behaviour dynamically (at run time) in response to transient changes in its execution environment or to permanent changes in its requirements [76]. To this end, the service needs to be informed about the networking environment in which it operates so that it can change its behaviour in order to provide the intended service despite the change in the environment.

Adaptation motivation: Context-aware adaptation was introduced in response to the highly-dynamic pervasive environment. This dynamism is cause by different factors: (i) in pervasive environments, the services and their parameters are subject of unpredictable changes: services may disappear; their behaviour, signature, or their quality-of-service characteristics may change over time. Consequently, as the composite service relies on services, faults and errors in the service execution maybe triggered. In this case, the service should recover from a faulty situation and return to normal operation (self-healing systems [77]). (ii) Different service consumers have different preferences, constraints, and QoS requirements. Therefore, the service should be able to dynamically adapt to these variations. (iii) The service should be able to adapt to the change in operating context. This includes for example the device context (e.g. memory available and physical dimensions), and the environment context (e.g. time, location, wireless signal loss, etc.). (iv) Service should be able to accommodate not only instance-level changes (for each consumer), but also the permanent behaviour change (evolution) of the service itself. This need could be motivated by, for example, the change of the business rules.

Adaptation Levels: Typically, the adaptation in the service (process) takes place in three levels: abstract level, service definition level, and instance level [78]. In some approaches the service model contains the service tasks in an abstract form. The adaptation in this level requires transforming the abstract service into a concrete one by determining the actual implementations of these tasks based on the available context before or during the service execution. The adaptation in the service definition level addresses the change in the business events or rules. It involves the modification of the service definition which should be propagated to the corresponding instances. The last level, instance level, takes place in the level of the instance of a concrete service definition and may include re-configuration or re-binding of the involved services according to the changes in QoS requirements. This thesis focuses on the adaptation in the service definition level.

In order for the developer (adaptation designer) to specify the required adaptation, usually two issues should be considered: (i) the moment the adaptation should take place (adaptation point) and (ii) how the adaptation should be performed (adaptation mechanisms). The adaptation point could be associated, for example, to a certain event such as reception of a message or time out. Such a point could also be associated to a certain application or environment state (context). In this case, it is defined as a complex condition regrouping relevant parameters. Finally, it could be associated to a specific control point in the business process model (e.g. executing a certain activity in BPEL). The next section describes the different adaptation mechanisms.

2.2.5 Adaptation Mechanisms

The existing adaptation approaches use different adaptation mechanisms which can be classified in three groups: goal-based, action-based, and variability-based [79].

Goal-based approaches

These approaches (e.g. [80]) define the goals to be reached by the system and the adaptation activities in a high-level form, leaving the system or the middleware to determine the concrete services at runtime to achieve the required goals based on some utility function. These approaches provide a degree of flexibility to define the adaptation actions. However, discovering options at runtime and making decisions depend on the expressiveness and completeness of service descriptions, and on the accuracy of the used decision making algorithm.

Action-based approaches

These approaches (e.g. [81]) rely on defining situation-action rules and therefore specify exactly what to do in every situation. Although it is easy and intuitive from the developer point of view, it may lead to a huge number of rules which may require analysis tools to identify the possible conflicting between these rules.

Variability-based approaches

These approaches (e.g. [20]) first identify the variation points in the service and its associated alternatives (variants) that specify different implementations or behaviours. Second, they specify variants selection mechanisms (based on ranking rules, preferences, etc.). These approaches enjoy the inherent power of a software product line in dealing with variability, automation and consistency.

Service modelling should be flexible enough to deal with constant changes – both at the business level (e.g. evolving business rules) and the technical level (e.g. platform upgrades). The flexibility could be provided or addressed by incorporating variabilities into a system [20]. The service adaptation is usually addressed (on the service instance or definition level) by explicitly specifying some form of variation points. To date, a variety of different adaptation approaches have been proposed for capturing variabilities (e.g. [20][82]). Common to all these approaches is that to differentiate between service family members they capture the service variant as a structure containing variation points. By making appropriate choices to resolve the variation points, either at design time or at runtime, a single service variant could be constructed. The proposed approach in this thesis can be classified in this variability-based group.

2.3 Current State of Enabling Technologies

2.3.1 Model Driven Architecture

Model Driven Development (MDD) is a software development approach that is based on the use of software modelling as a primary form of expression [42]. Software models are constructed, and then code is written by hand in a separate step. Alternatively, complete software models are built including executable actions. Code can be automatically generated from the models, ranging from system skeletons to complete, deployable products. MDD has become very popular today especially after the introduction of the Unified Modelling Language (UML). Later, with the increased focus on architecture and automation MDD technologies have evolved to provide higher levels of abstraction in software development which promotes models with a greater focus on problem space. Therefore, the Object Management Group (OMG) has developed a set of standards called Model Driven Architecture (MDA), building a foundation for this advanced architecture-focused approach.

The Model Driven Architecture (MDA) emphasizes the use of models throughout the application development lifecycle [83]. It aims to provide a system for complete cycle of analysis, design, and implementation of applications. All MDA development projects start with a Platform Independent Model (PIM), which is expressed in UML. The base PIM is then (semi)automatically transformed to a Platform Specific Model (PSM) using some transformation tool and possibly with some additional information that guides the transformation process [84]. This transformation allows for higher run-time performance through automated optimizations not feasible with handwritten code.

Due to the platform independence MDA can be used with CORBA, COM, Java, C#/.NET, XML/SOAP and any future middleware software [85]. UML allows an application model to be constructed, viewed, developed, and manipulated in a standard way at analysis and design time. Just as blueprints represent the design for an office building, UML models represent the design for an application, allowing business functionality and behaviour to be represented clearly by business experts at the first stage of development. This allows the design to be evaluated when changes are easiest and least expensive to make, before it is coded.

In pervasive environment, for example, there exist different types of embedded devices with varying capabilities and requirements. Developing applications for these devices is a difficult task to the programmer as it involves low-level embedded knowledge together with domain expertise. MDA allows a PIM (which captures the high-level design) and multiple PSMs (which capture implementation and platform-specific details of each device) to be defined.

2.3.2 Process Mining

Process mining techniques use log data to analyze observed processes and have been successfully applied to real-life logs from, e.g., hospitals, banks, etc. [86]. The basic idea of process mining is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs. The activities occurring in processes are either supported or monitored by information systems. However, process mining is not limited to information systems and can also be used to monitor other processes [87]. The common denominator in the various applications of process mining is that there is a notion of a process and that the occurrences of activities are recorded in so-called event logs [88]. Process models are structures that model behaviour. Although the idea of process mining is related to some work discussed in the machine learning domain, the targeted process models reside at the net level (e.g., Petri nets) rather than sequential or lower level representations (e.g., Markov chains, finite state machines) [89]. Therefore, process mining needs to deal with various forms of concurrency. Moreover, as will be seen later the process could be analyzed not only from the control flow perspective but from different perspectives.

In the area of process mining, there are different algorithmic approaches, which derive the control-flow and other models (e.g. the organization and the information models) from the data logs [90]. These algorithms are integrated as plug-ins in the ProM tool [91]. This thesis focuses on control flow mining algorithms to understand the user behaviour and recognize her contextual situation as will be seen in Chapter 6.

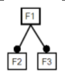
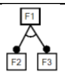
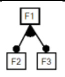
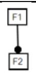
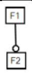
2.3.3 Software Product Line

The commonality and variability management techniques from Software Product Line (SPL) are appealing because as will be seen in Chapter 4 it can be applied to handle context variabilities. According to [92] a SPL is "a set of features that satisfy specific needs of a particular market or mission, and that are developed from a common set of core assets in a prescribed way". Thus, SPL is an effective approach to software development that promotes "reuse" to a first-class entity aiming at reducing overall development time and cost while improving product quality.

A product line architecture represents an architectural structure for a set of related products by defining core elements that are present in all product architectures, and variation points where differences might occur among specific product architectures. Each variation point is guarded with a Boolean expression. Given a set of desired properties or bindings, a particular product architecture can be selected out of a product line architecture by resolving the Boolean guards of each variation point. Treating software as a product line is a new approach to support software variability from design-time to invocation-time to run-time [93].

The feature model proposed in [94] has generated a lot of interest in the SPL community. By modelling a product hierarchy of features with their similarities, differences and relationships, feature model plays an important role in SPLs. It provides means to represent the commonalities and variabilities within a family of systems which allows individual family members to be safely configured. Commonly there are five types of relations possible in a feature model [13] (see Table 2.1). Additional constraints between features may exist that describe how features interact with each other e.g. *requires* and *excludes* constraints. These relationships and constraints have been used to model the dependencies between context features as will be seen in Chapter 4.

Table 2.1 Feature type relations

And: if F1 is selected, subfeatures (F2,F3) must be part of any product of the product line	
Alternative: if F1 is selected, only one subfeature (F2 or F3) can be selected in any product in the product line.	
Or: if F1 is selected, one or more subfeatures can be selected as part of any product in the product line.	
Mandatory: if F1 is selected, the subfeature is required as part of any product in the product line.	
Optional: if F1 is selected, the subfeature may or may not be part of a product in the product line.	

2.3.4 Jabber Overview

The collaboration between different context servers distributed in different domains requires generic APIs and an appropriate communication protocol allowing context information exchange between different entities: context servers, context providers, and context consumers. Relying on a standard protocol is obviously a preferred choice. Jabber is an extensible instant messaging (IM) system. More precisely, Jabber is a set of streaming XML protocols and technologies that enable any two entities on the Internet to exchange messages, presence, and any other structured information in near real-time.

The Internet Engineering Task Force (IETF) has standardized the core Jabber protocol as the XMPP protocol [95]. The architecture of the Jabber system is distributed. A Jabber server has a number of registered clients. Clients on the same server interact through that server; clients on different servers interact through server-to-server communication. Jabber enables message transfer not only between people, as in traditional Instant Messaging (IM) systems, but also between any two entities. An entity can be a person, a device, or a software service. Each entity has a unique Jabber ID (JID). A JID is similar to an e-mail address. For example, a JID for Alice is `Alice@merchiston.napier.ac.uk`.

Furthermore, Jabber enriches the communication support beyond chat to many other interaction semantics thanks to the XMPP extensions. The Jabber Software Foundation develops extensions to XMPP through a standards process centred on XMPP Extension Protocols (XEPs) [96]. Examples of these extensions are the Jabber RPC [XEP-0009], ad-hoc commands [XEP-0050], streaming audio and video [XEP-0166], and so on.

In addition, Jabber has an interesting pubsub facility [XEP-0060], in which both publishers and subscribers are Jabber entities. A publisher publishes a message item to a topic, and then all the topic subscribers will be notified about the newly published item. In this communication mechanism, since the publisher does not know who will receive the message, and a subscriber does not know who sent it, the time-coupling and reference-decoupling between publishers and subscribers are assured. This pubsub mechanism is ideal for implementing *ubique* middleware as will be seen, where context providers and consumers can be associated and disassociated dynamically.

2.4 Conclusions

Based on the literature review, the following conclusions have been reached:

1- Ontologies are a very promising instrument for modelling contextual information due to their high and formal expressiveness and the possibilities for applying ontology reasoning techniques. Thus, this thesis focuses on context management employing ontologies as the underlying technology.

2- In a pervasive environment, the context manger is expected to administer a large volume of context information represented by RDF triples in the context repository. Applying the reasoning capability to infer new context knowledge may have a severe impact on the overall performance of the system. That is, for any new event or context information added to the repository there is a chance of deducing new context knowledge by applying ontology and rule-based reasoning. Therefore, applying ontology-based modelling is still inefficient due to performance limitations. Thus, there is a need to improve the reasoning performance and reduce the response time which is a vital issue in a pervasive environment.

3- The SPL is a promising technology to model context variability as will be seen in Chapter 4. Therefore, in order for the middleware to serve different types of applications, SPL could be leveraged to provide context-specific programming abstraction or constructs that model the context variability.

4- In pervasive environments, context management systems are expecting to administer contextual information which is naturally originated from different domains (areas). Each domain may maintain its own sensors and mechanisms for inferring context. Thus, a cross-domain context management and collaboration framework is needed. In particular, the design of distributed storage, retrieval, and dissemination mechanisms of context information is vital.

5- The communication benefits of Jabber technology can be leveraged to design robust and scalable middleware architecture for distributed context managements and cross-domain context information dissemination. The nature of this technology makes it a potentially suitable ingredient of *ubique*, the proposed middleware architecture to distributed context management.

6- Contextual situations (high-level context information derived from low-level sensor readings) are more stable and easier to define and manage than basic context information. Thus, situation awareness is needed to allow the entities in pervasive computing environments to be aware of situation changes and automatically adapt themselves to such changes to satisfy user requirements.

7- Because context information is naturally distributed in different domains (areas), recognizing user's situations among the flow of context information may require considering not only the context information history but also the states the user experienced in these domains.

8- Context-awareness and adaptability are important and desirable properties of services to provide users with personalized offering. In addition, service modelling must be flexible enough to deal with constant changes. It is promising to provide or address this flexibility by incorporating variabilities in a logical way so that the developer can view the service variant as a set of features that determine the difference between service variants in each usage context.

9- As the service engineering process passes through the stages of analysis and design prior to the actual code development, the context and adaptation should be considered also in these stages. In this respect, it is promising to develop a new method to automatically derive the service variant based on the current context by leveraging model driven development and generative programming techniques.

Chapter 3 Related Work

3.1 Context Modelling

3.1.1 Requirements of Context Modelling

A number of ontologies have been developed specifically for use in pervasive computing such as CoBrA [31], Gaia [97], CoOL [98], CONON [33], GAS [99], and CoDAMoS [100]. In the literature, there exist different surveys on context modelling approaches (e.g., [37][101][9][102][103]); each of them evaluates the different context models with respect to different criteria. For example, the authors of [102] compare and evaluate the above mentioned most popular ontologies against the system challenges generally recognized within the pervasive computing community and with respect to ontology-modelling best practices.

However, the investigation that forms the basis of this dissertation addresses issues related to providing context designers and developers with methodologies and tools for developing context-aware services which facilitates their task. For this aim, the evaluation of ontology-based approaches to model context is addressed from the perspective of their efficacy in meeting the requirements of a straightforward way of developing context-aware applications and efficient applicability of ontologies to context modelling. To this end, this dissertation adds to the general criteria mentioned in [57][101][9][104] different criteria derived from the literature and from the author's own experience in developing such applications.

The following shows the requirements (R1-R5) for context modelling technique that can be used to evaluate the ontology-based context techniques in Section 3.1.3. It may not be possible for a modelling technique to fulfil all requirements. Moreover, there is no clear indication which requirement has priority.

R1- Efficient applicability of context reasoning: As aforementioned, in a pervasive environment, the context manager is expected to manage a large volume of context information represented by RDF triples in the context repository. Thus applying the reasoning capability to infer new context knowledge may have a severe impact on the overall performance of the system. Therefore, techniques for pre-selection of context information relevant to an application, which could speed up the reasoning process by reducing the size of the knowledge base, are needed.

R2- Ease of context querying: applications use context queries to retrieve the set of context information that adheres to some conditions. Some context queries are difficult to be defined using general-purpose querying mechanisms (e.g., SPARQL). In addition, the application developer may not have enough knowledge about the context semantics, in order to describe queries correctly.

R3- Providing different levels of abstractions: The context model should provide context information in arbitrary levels of abstraction. It should hide irrelevant context details and offer a high-level interpretation of lower-level context details. The level of details should be specified by the application.

R4- Efficient context provisioning: in the presence of large models and numerous data objects, efficient access to context information becomes a requirement [9]. In order for the applications to access the relevant context information suitable access paths have to be represented in the context modelling. These access paths define the primary dimensions which will be used to access the secondary context. For example, primary context attributes could be the object identity, its location or activity, etc. The context modelling technique should provide mechanisms that allow different applications to express their access paths according to their needs. Further, to effectively provide different applications with relevant context information it may be necessary to find a mechanism to decrease the number of network interactions between an application and the context provider which may improve the overall performance of the system.

R5- Provide constructs to model context variability: in order for the context management system to serve different types of applications, it should provide context-specific programming abstractions or constructs that model the context variability. Therefore, the context modelling technique should provide application developers with software constructs through which a view-based customization of the context knowledge could be expressed.

3.1.2 Context Modelling Approaches

3.1.2.1 Context Model of CoBrA

In CoBrA [31], a broker-centric agent-based architecture for supporting context-aware computing in intelligent spaces, contextual information is represented by a set of ontologies called COBRA-ONT that is implemented in OWL [61]. CoBrA-ONT defines typical concepts and relations for describing physical locations, time, people, software agents, mobile devices, and meeting events, which supports smart meeting applications. Subsequently, a set of more general ontologies, named SOUPA [105] (Standard Ontology for Ubiquitous and Pervasive Applications), has been proposed for supporting pervasive computing applications.

SOUPA organizes its ontologies into SOUPA core and extension. The SOUPA Core ontologies define generic vocabularies (including Person, Agent, Event, Space, Time, Action, and Policy) that are universal for different pervasive computing applications. By extending the core ontologies, the SOUPA Extension ontologies define task-dependent vocabularies for supporting specific types of applications.

SOUPA offers a formal and well-structured way to model context, and thus provides rich semantics for programming. It also allows policies to be defined to support trust and privacy. This is demonstrated in CoBrA's EasyMeeting application [106], in which the ontologies facilitate knowledge sharing and work with logic inference rules to reason about the context to infer new context knowledge (e.g., spatial relations, device profiles) that cannot be easily acquired from the physical sensors. Thus, by using these rules, different levels of context abstractions could be achieved; however, these rules are specified independently of the applications' needs. Moreover, application developers should have enough understanding of the internal structure and semantics of the context information in order to specify the required queries. Finally, the rigid reasoning schema does not permit different reasoning schemas for different applications' needs, i.e. context variability is not addressed.

3.1.2.2 Context Model of Gaia

In Gaia [1], an infrastructure for smart spaces, ontologies are introduced to provide a standard taxonomy of the different kinds of entities (including applications, services, devices, users, and data sources). Therefore, these ontologies are beneficial for semantic discovery and interoperability between entities. Additionally, the Gaia ontologies are used to make Gaia systems context-aware. They model contextual information including physical, environmental, personal, social, and system contexts.

The Gaia context model is based on first-order logic and Boolean algebra, which permits easily written rules to describe context information. An atomic context predicate is defined as `Context(<ContextType>, <Subject>, <Relater>, <Object>)`. It is written in DAML+OIL [107]. More complex contexts can be constructed by performing first-order logic operations such as quantification, implication, conjunction, disjunction, and context predicate negation.

Moreover, to present context as directories, Gaia introduced the *context file system* to construct a virtual directory hierarchy, based on the types of context associated with particular files, in which path components represent context types and values. This virtual directory hierarchy forms a simple query language to determine what types of context are attached to files. For example, to determine which files have the associated context: `location = Room3 And situation = meeting`, we enter `/location:/Room3/situation:/meeting` directory. Therefore, develops can easily query the context repository. In addition, by using the context file system primary dimensions, which will be used to access the secondary context, can be easily identified and mapped to file paths. However, reasoning about available context information from different perspectives is neglected.

3.1.2.3 ASC Context Model

Aspect-Scale-Context (ASC) is a model for describing contexts and their relationships using ontologies as fundamental [108]. A context is a set of *ContextInformation* instances characterizing entities (like a person, place, or a general object) relevant for a specific task in their relevant aspects. These instances are defined and interlinked by use of the aspect-scale-context (ASC) model. An Aspect is a classification whose subsets are a super-set of all reachable states, grouped in one or more related dimensions called Scales. A Scale specifies fine-grained representation formats for an aspect, for example, a distance aspect has multiple scales such as metre, kilometre, and nautical mile. The ASC model shows how contextual information may be used to characterize a state of an entity under a specific aspect.

CoOL, the Context Ontology Language [108], is derived from ASC to facilitate ontology-based contextual interoperability. CoOL is divided into two subsets: (i) the *CoOL Core*, which projects ASC model into various common ontology languages such as OWL and DAML+OIL, and F-Logic [109]; and (ii) *CoOL Integration*, which is a collection of schema and protocol extensions as well as common sub-concepts of ASC. CoOL is used to enable context interoperability and context-awareness during service discovery and execution. By using rules and an inference engine, the context provider is able to derive new knowledge from CoOL-based knowledge, and to validate ontology consistency. In fact, using the aspect concept facilitates the definition of the context access paths to effectively provide context information.

However, because the inference is done on monolithic CoOL-based knowledge, the context reasoning may be inefficient. In addition, the CoOL model partially supports context variability. However, it is not generic enough to model the aspects hierarchy and their dependency. For example, if we consider the *Publication* as an aspect for a *Researcher* object, we may have different sub-aspects (e.g. conferences, journals, and book chapters); each of which has different scales. Thus, a more generic approach to model the context variability is needed. Finally, CoOL is less practical for expressing aspects' scales with regards to more non-material context data, such as user preference or activity.

3.1.2.4 Context Model of SOCAM

The CONtext ONtology (CONON) is an ontology-based context model, in which a hierarchical approach is adopted for designing context ontologies [33]. Contexts are represented as predicates written in OWL. CONON is used in the Service-Oriented Context-Aware Middleware (SOCAM) [36], an architecture that enables the building and rapid prototyping of context-aware services in pervasive computing environments.

Similar to ULCO [110] and COMANTO [111] ontologies, CONON includes a common upper ontology that captures general concepts about basic context in pervasive computing (such as person, location, computing entity, and activity), and also provides the possibility of defining a domain specific context model (e.g., smart homes) by extending the upper ontology for adding domain-specific ontology in a hierarchical manner. Domain-specific ontologies can be dynamically "bounded" or "re-bounded" with the upper ontology when the domain is changed. For example, when a user leaves his home to drive a car, the home-domain ontology will be automatically replaced by the vehicle-domain ontology.

To support various kinds of reasoning tasks, multiple logic reasoners are considered: RDFS reasoner, OWL reasoner and a general rule-based reasoner. Therefore, different levels of abstractions could be achieved but these levels are not specified by the applications. A context-aware home scenario is implemented in the prototype system to demonstrate the use of CONON [112].

By tailoring the upper context ontology and domain-specific ontologies in the context model, context reasoner has a reasonable performance over small-scale context knowledge in pervasive environments [36]. However, CONON does not provide constructs to model context variability to support different applications with different reasoning schema needs.

3.1.2.5 Context Model of ACAI

In [113] the authors have designed context ontology adequate for supporting their ACAI architecture (Agent-based Context Aware Infrastructure). They argue that instead of modelling context according to its functional intentions, context should be modelled according to the tasks the application layer performs. Therefore, they decided to model context in several levels of expressiveness, where the highest level is an abstraction of all concepts of context. When going down through the levels, context is expressed in more detail and refined more concretely. The highest level of abstraction is the *ContextView* which represents the different types of context that belong to a given entity. Thus *ContextView* represents the primary dimension which will be used to access the secondary context. *ContextView* has two properties *contains*, and *invokes*. The classes *ContextFeatures* and *ContextEngagements* are the respective ranges of those properties. These classes are considered to be the second level of expressiveness in the ontology.

In order to be able to reason about the available context information, Khedr et al. developed a relational and dependency ontology model and implemented an inference engine in order to derive logical, social and composable context. The dependency ontology has a wide range of predicates that correspond to the different *ContextFeatures* represented. The ontology consists of five rule-type categories: *ActionDependency*, *ActorDependency*, *LocationDependency*, *ServiceDependency*, and *RoleDependency*. Therefore, this separation of predicates permits efficient application of the context reasoning by considering only the dependencies needed by an application. However, ACAI ontology is rather elementary with regards to the context features and types defined. In addition, a more generic solution is still needed which does not impose any restriction either on the number of these dependencies or on the number of context features.

3.1.2.6 The MUSIC context modelling

The MUSIC context model is a result of a research project called *Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environments (MUSIC)* [114]. The goal of MUSIC is to develop an open-source computing infrastructure and an associated software development methodology that facilitate the development of self-adapting, context-aware applications in pervasive environments. The proposed context model follows an ontology-based approach and has three layers of abstraction, i.e. conceptual layer, exchange layer, and functional layer [115]. These three layers facilitate the analysis and design of context-aware applications as part of a comprehensive, model-driven software engineering process.

The MUSIC context model is inspired by and complements the ASC model with MDD support. The conceptual layer aims to be leveraged by the developers and to be exploited in the MDD approach. In this layer, the ontology is described in OWL and the context meta-model is specified in UML. At the exchange layer, an instance of the conceptual model is represented in e.g. XML. The functional layer also defines a set of data structures for storing the context information.

Context querying is facilitated by providing the developer with a Context Query Language (CQL) [116]. In addition, similar to SOCAM, MUSIC provides an ontology that is divided into two corresponding hierarchies: concepts and representations. This division allows the use of only the light-weight concepts hierarchy for context reasoning while omitting large parts of the ontology that only contain the representations; thus rendering relatively efficient context reasoning. However, this model does not provide the developer with any programming constructs used to express the level of abstraction required for an application. Moreover, the context variability is not supported in this model.

3.1.3 Evaluation of the Context Modelling Approaches

In Table 3.1, the surveyed approaches to model the context information are listed. All these approaches are evaluated on how well each approach fulfils the context modelling requirements specified in Section 3.1.1. None of these approaches appears to cover adequately the space of concerns defined by these requirements.

Table 3.1 Requirements for context modelling techniques

Context Model	R1	R2	R3	R4	R5
Context Model of CoBrA	-	~	-	+	-
Context Model of Gaia	-	+	-	++	-
ASC Context Model	-	~	-	+	~
Context Model of SOCAM	+	~	-	+	-
Context Model of ACAI	+	~	+	++	-
MUSIC Context Model	+	+	-	+	-
+ fulfilled. - not fulfilled. ~ partially fulfilled					

R1- Efficient applicability of context reasoning

R2- Ease of context querying

R3- Providing different levels of abstractions
R4- Efficient context provisioning
R5- Provide constructs to model context variability

As shown from the comparison, existing context modelling approaches address a sub-set of these challenges only, or cover some of them only to a limited extent. Moreover, most of them view context modelling either from a purely conceptual or a purely functional perspective. However, when engineering context-aware systems two main aspects should be addressed at the same time: defining the semantics and relations between context elements at a conceptual view, and providing context constructs that can serve different applications. Thus, this thesis presents an attempt to facilitate the developer task in dealing with these aspects.

3.2 Context Management Architectures

3.2.1 Driving Requirements

Hereafter, we refer to the computational entity responsible for transparently binding the context consumers (CCs) (i.e. applications) with corresponding context providers (CPs) a context server (CS). The context management in each domain is done by the CS available in that domain. The complexity of developing context-aware applications that require context information available in different CSs makes the use of a context management middleware crucial. This middleware should address many of the requirements of traditional distributed systems, such as heterogeneity, mobility, and scalability. In addition, it should fulfil other key requirements such as:

Domains of context perception: Since the context information is naturally distributed, the context management must be distributed in order to allow efficient and scalable dissemination of context. However, the task of context-aware developers becomes more difficult as it requires a priori knowledge of the computational entities responsible for providing the context information they are interested in. Their task becomes even more complex when context providers dynamically enter and leave the pervasive environment. Thus, there is a need for a dynamic discovery mechanism of context providers.

Furthermore, the middleware scalability could be increased by restricting the access and perception of the context to some domains [29]. This requirement conforms to the principle of system boundary [117] of pervasive applications.

Uniform API interface and protocol: In order to enable every party to become a context provider and implement its own CS, every CS should: (i) obey a certain protocol with which context information can be disseminated between different CSs; and (ii) implement a standard API which allows context providers to register and publish context information in it, and context consumers to acquire context information they are interested in. This way, for instance, an organization can operate a CS for its members, and an individual can run a CS as a context provider for a single user or family members. Therefore, similar to the Next Generation Service Interfaces (NGSI) [118], providing a standard API for accessing such information, allows third party application developers to build new services based on the context made available to them.

Efficient context information dissemination: With regard to situations involving mobile users roaming across domains, additional restrictions may arise (e.g. concerning limited connectivity and bandwidth, unknown network conditions, etc.), thus exchanging context information between domains should be fast and only the required information should be transferred when users roam across domains. This requirement calls for a dissemination protocol between CSs. Furthermore, the middleware should support the “publish on demand” mode of operation. That is, if a context provider publishes at a higher rate the context information is more accurate in terms of freshness. However, this is a costly operation in terms of the network bandwidth usage, processing power, and energy consumption (e.g. battery usage of WiFi scanners). Thus, the middleware should enable providers to publish when there is a corresponding consumer and according to the publishing rate needed by the consumer.

Cross-domain reasoning: As the context information is originated from different domains, the context management system should support reasoning about context information spanning multiple domains. That is, in order to track user's behaviour there is a need to consider the context information available in the different domains the user visits [119]. Hence, understanding the user's current situation may require considering the different states the user experienced in these domains. For example, to identify if the current day was busy for the user there is a need to consider the different activities and states the user has experienced in work, shopping, on the road, etc.

Dynamic matching between context providers and consumers: Typically developers define context interests (queries) which should be transparently kept across distributed CSs. The middleware should allow the context consumers (applications) to register their interests in context information; and the context providers to register their capabilities. Then, for any change in either the context consumers or providers, a matching function should be triggered so that applications asynchronously receive notifications of context information that match their interests. In addition, the application should be able to specify its context interests on the basis of context types and meta-attributes such as precision and accuracy and to indicate additional restrictions based on properties of the provider or the context publication. In this case, the middleware has to choose the most adequate context providers among the available dynamic set of providers.

Support for privacy: The flow of context information between different distributed domains obviously raises user privacy issue. A context-aware echo-system should protect user’s information and guarantee privacy across domains.

3.2.2 Existing Context Management Architectures

Classical work in context-aware computing has developed centralized and application-specific solutions such as Context Toolkit [32] which provides a set of abstractions that can be used to implement reusable software components for context sensing and interpretation. The context information is directly acquired from a sensor by means of the context *widget* component. Widgets can be combined with *interpreters*, which transform low-level information into higher-level information that is more useful to applications, and *aggregators*, which group related context information together in a single component. Finally, context-aware applications can invoke actions using *actuators*, and locate suitable widgets, interpreters, and aggregators using discoverers. Another interesting study is Gaia [1] which adopts the concept of *active spaces*. Active spaces are physical spaces where devices in a heterogeneous network, such as PDAs and printers, can discover each other, auto-configure and dynamically start a context-aware interaction. It provides a framework to develop user-centric, resource-aware, multi-device, context-sensitive and mobile applications. However, these approaches offer solutions for restricted and small-size smart space environments, with localized scalability.

More recent middleware offer access to context information in distributed repositories. For example, the Context Fabric (Confab) [4] provides architecture for privacy-sensitive systems, as well as a set of privacy mechanisms that can be used by application developers. It maintains context information in distributed tuple-spaces called *infospaces*. Each infospace is a repository responsible for storing one or more context types. An application interested in a certain context, builds a context query using the address of the responsible infospace. In order to handle queries over distributed infospaces, Confab offers a query processing service, which distributes queries over distributed infospaces and composes the query results. Privacy is supported by adding operators to an infospace to carry out actions when tuples enter or leave the space. However, Confab does not adequately address the other middleware requirements such as mobility or context information dissemination across domains.

The scalability issue is considered in PACE [120], which is another distributed middleware focusing on offering a context model called CML (Context Modelling Language) and advanced context-based programming abstractions for distributed context-aware applications. PACE is organized in layers that provide, in addition to context management, an interface to execute distributed context queries, and an adaptation layer, which maintains a reusable repository of adaptation abstractions. Applications use a catalogue and meta-attributes to discover which repository satisfies their context requirements. However, this discovery mechanism does not allow the developers to identify the context repositories (CSs) existing in the domains visited by the roaming users and holding their context information.

CAMUS [5] is another distributed middleware where context-aware system federation is composed by environments based on CAMUS services, which disseminate context information as tuples. Each service of an environment must be registered in a Jini discovery service. A CAMUS context domain is an environment that supports a minimum set of CAMUS services. The set of Jini services responsible for each CAMUS domain composes a federation. In order to access context information or to use a service of a specific domain, a client must query the Jini federation, using parameters such as the name and localization of the domain. CAMUS, however, does not address cross-domain context dissemination and how to ensure user’s privacy.

Another interesting approach to allowing distributed context management based on federating context-aware services is Nexus [3]. Nexus supports heterogeneity among context management systems' context models, i.e. each context management system can adopt a particular context model and must implement an abstract interface and register itself at an *Area Service Register*. Thus, it focuses on the data management aspect of large-scale pervasive computing systems. A client may access context information provided by the federation, by using a query language. However, there is no concept such as domain or environment: each context server is a repository of a specific context type [29]. Similar to Nexus, GLOSS [6] composes heterogeneous context management systems through hierarchical or peer-to-peer interconnection methods. By introducing the notion of *Global Smart Spaces*, GLOSS supports interaction amongst people, artefacts and places while taking into account both context and movement on a global scale that facilitates the implementation of location–aware services. It allows users to pick up small notes left for them in the environment. GLOSS uses the idea of home nodes used in the proposed approach in this thesis, however, it has been designed to manage location context only.

Compared to this approach, Chen et al. [7] propose *Solar*, a data-centric infrastructure based on Context Fusion Networks (CFNs) to support context-aware pervasive-computing applications. CFNs are based on an operator graph model, in which context processing is specified by application developers in terms of sources, sinks and channels. In this model, sensors are represented by sources, and applications by sinks. Operators, which are responsible for data processing, act as both sources and sinks. At runtime, the implemented peer-to-peer (P2P) infrastructure instantiates the operator graphs on behalf of context-aware applications. *Solar* consists of a set of functionally equivalent hosts named *Planets*. The components messages will be delivered to a *Planet* with the numerically closest ID; therefore, unlike the proposed approach in this thesis, *Solar* services focus on the data objects instead of on where they live i.e. from which domain they originate. In addition, *Solar* does not address privacy enforcement.

Another hybrid approach to modelling contextual information that incorporates the advantages of object-oriented and ontology-based modelling techniques is introduced by Lee and Meier [121]. The objective is to support a specific large-scale pervasive domain, namely the transportation domain. Their notion of Primary-Context Model and the Primary-Context Ontology is used to share context between different domains. Although their approach is interesting, it does not address other issues such as mobility and cross-domain context dissemination.

ICE [122] is a scalable context management middleware for Next Generation Networks. It is based on the concepts of *context sessions* and *context flows*. The idea is to separate signalling data from content exchange, as in IP Multimedia Subsystem, to establish context sessions for more scalable and adaptive management of context information. The Context Access Language (CALA) has been designed to support context queries and subscriptions. However, ICE focuses heavily on efficient context information dissemination between context sources and sinks, and ignores in its designed protocols ensuring entities privacy. In addition, ICE requires that the context sources' descriptions and the context sinks' queries/subscriptions to be registered in a centralized entity - the context broker. Therefore, as the user roams between domains, this adds complexity to the developers as they must know in advance which context broker has the context information they are interested in.

The Context Management Framework (CMF) proposed in the MobiLife project [123][124] is designed for the discovery of, exchange of, and reasoning on context information. It is a set of components, which are connected at run time, that together provide the relevant context information for the service or application, using sensing and interpretation mechanisms. The main tasks for the CMF are to enable the discovery of context providers, to provide a published agreement or interface contract between context providers and context consumers, and binding context consumers with the matched context providers in order to use their context service functions through the use of a context broker. Therefore, in CMF there is no concept such as domain so that the application is able to specify the domain(s) from which the context information is originated. In addition, the infrastructure needed for setting and enforcing privacy of user-controlled data available through context providers is controlled by the Trust Engine. However, this thesis argues that this setting weakens enforcing the privacy since a malicious context provider can skip contacting the trust engine to verify if the context consumer is eligible to access the context information.

Zebedee et al. [125] introduced ACMF, an adaptable context management system by adopting autonomic computing paradigm. ACMF is implemented by using the Web services and the Web Services Distributed Management (WSDM) standards. ACMF views each device in terms of the roles it plays with respect to context management which includes client, server, and context proxy. ACMF defines a context model and a set of context exchange protocols between devices. ACMF models the pervasive computing environment as a collection of domains where each domain contains a set of regions and a set of device types. A domain is a logical representation of a physical space, such as a building or campus, containing regions and device-types. In this respect, their domain concept is similar to the domain concept used in the proposed approach in this thesis. However, because the focus is on exchanging context information between devices available on a local area (one region) ACMF does not address cross-domain context dissemination, which is a requirement in a pervasive environment. Therefore, querying context information available in distributed domains is not possible in their approach.

From the perspective of globally connecting sensors, the Open Geospatial Consortium provided the Sensor Web Enablement (SWE) initiative [126] to build a framework of open standards for exploiting Web-connected sensors and sensor systems of all types such as flood gauges, air pollution monitors, Webcams, etc. SWE provides the opportunity for adding a real-time sensor dimension to the Internet and the Web. It focuses on developing standards to enable the discovery, exchange, and processing of sensor observations, as well as the tasking of sensor systems in order to achieve a "plug-and-play" Web-based sensor networks. Thus, SWE cannot be directly applied to achieve context-awareness because, for example, Sensor Model Language (SensorML) describes sensors systems and provides information needed for the discovery of sensors, the location of sensor observations, etc. But it does not consider modelling the entities about which the sensor is able to provide information.

Most of the previous work focussed on the software engineering perspective of the distributed context management. Castelli and Zambonelli [127] addressed the distributed management of context information from a knowledge management perspective. They propose a self-organized agent-based approach to autonomously organize distributed contextual data items into knowledge networks. These data atoms as well as any higher-level piece of contextual knowledge represents a fact which can be expressed by means of a four-field tuple (Who, What, Where, When); they call it the W4 Data Model. This model is able to represent data coming from heterogeneous sources and to promote ease of management and processing. These knowledge atoms are linked via general-purpose mechanisms and policies to form W4 knowledge networks which can facilitate services in extracting useful information out of a large amount of distributed contextual items. The usage of tuple-space like repositories supports heterogeneity and facilitates building the knowledge network. However, because the focus is on the knowledge management perspective other requirements such as mobility between domains have been partially addressed. In addition, despite the efficiency in retrieving tuples during the query resolution phase, using the spidering approach to create the knowledge networks may be inefficient when considering the rapidly changing context information such as entities location.

If we look at the aforementioned requirements and at the approaches described above, it reveals that research in the area of context management is well established and many ideas have been developed for addressing most of the above requirements individually. However, none of the examined approaches supports all of our requirements to a sufficient extent. Therefore, there is a need to design a new context management infrastructure that takes into consideration the distribution of context in different domains and the necessity to ensure user privacy.

3.3 Situation Recognition Approaches

Situations can be recognised and learned automatically by aggregating sensor readings and associating them to a human-defined situation label using for example machine learning techniques. Alternatively, situation models can be defined manually by domain experts. Thus, situation recognition approaches can be roughly grouped into three categories: specification-based approaches, machine learning based approaches, and a combination of both.

3.3.1 Specification-Based Approaches

The common denominator of these approaches is that they define the situation as a set of rules and try to find an exact match in the context information. These approaches are suitable when contextual situation ingredients are known in advance; in this case an expert can specify the situations manually based on his knowledge. These approaches can be subsequently classified into rule-based and ontology-based.

Rule-based approaches

Early approaches use formal logic and temporal logic to describe and represent situations. For example, Loke [10] views the situation as a set of relations between objects, thus recognizing a situation boils down to determining if a prescribed set of such relations hold or do not hold at that given point in time. In his approach situations are represented within a logic programming language and manipulated as first-class entities. Therefore, a situation is defined as a collection of rules (or a logic program), which is called a situation program. The rules of a situation program permit natural representation of a situation, i.e. if a situation occurs, then certain conditions and constraints should hold. Loke described six different ways to specify the situation *in_meeting_now* based on different contextual facts.

Another interesting example is Gaia project [11] where the model of context is based on first order predicate calculus, and the reasoning about high-level context is based on the pre-defined rules. For example, what kind of activity is going on in the room can be recognized based on the number of people in the room and the applications running in the room.

The main limitation of logic-based approaches is that they are not suitable for inference from imprecise and incomplete contexts as they are designed for exact reasoning. In addition, they do not take semantics into consideration.

Ontology-based approaches

Ontology-based reasoning approaches incorporate the semantics into context representation and reasoning [33]. Below are some examples of these approaches.

In [59], Springer et al. presented an interesting approach which provides a conceptual architecture and generic framework that enables an easy and flexible development of situation-aware systems. This architecture covers the whole process of context capturing, context abstraction and decision making. To handle complex situations the concept of decomposition is applied to the situation to achieve a hierarchy of sub-situations. However, a more generic approach is needed so that defining situations in terms of states should not only consider logical relationships but also temporal and dependency ones.

In [12], OWL-based situation ontology is presented to model situation hierarchically to facilitate sharing and reusing of situation knowledge and logic inferences. This situation ontology models the upper ontology for context and situations in pervasive computing environments using OWL-DL which can be easily extended in each domain and facilitates the sharing and reusing of situation information. However, they follow the traditional scheme of identifying situations by using logic reasoning which involves exact matching with a specified situation model.

Ontology-based approaches have limited capability in dynamically inferring contexts. It requires defining all the rules beforehand, and that all ontologies related to the specific domain must be defined already [128]. Due to a lack of comprehensive knowledge about their domains, users have to resort to domain experts; this leads to higher human cost and restricts the ontology application.

Other approaches focus on defining and modelling the relationships between situations. The rationale behind these approaches is, given a current situation and its relations to other situations, the search space for potential situations to be recognised is reduced. For example Reignier et al. [62] represent situations relationships by Allen’s temporal logic. These temporal relations are compiled into a Petri net that takes contextual changes as input to trigger the situation transitions. They emphasise the constraint that at least one situation must be active at a time which provides more stability and better performance. However, this requires the developer to design a complete situation model covering all potential situations, their relationships and transitions which is not always possible. Thus, the developer task becomes more difficult.

Generally speaking, the rule-based and ontology-based approaches provide the flexibility to represent a situation in multiple ways. In addition, the modularity of representing situations emphasizes the incremental approach and reuse when building a knowledge base of situations. However, in the domain of context-aware computing, these approaches are error-prone due to the incompleteness and ambiguity of context information. Furthermore, limited reasoning performance reduces its applicability in real world applications.

3.3.2 Machine Learning Based Approaches

In these approaches situations are recognized automatically by aggregating the sensor readings using one of the machine learning techniques. In the learning phase, a specified set of sensor readings values are associated to a human-defines situation labels.

The learning based situation recognition is related to the domain of human activity recognition. Typically the existing approaches to activity detection require constructing sequence-based models of low-level activity features based on the order of object usage. However, Palmes et al. [17] argue that activities may have a distinct series of steps but with no particular sequence. Thus, relying on sequence of events for activity recognition may significantly limit the accuracy and applicability of models that rely particularly on object sequence. Therefore, they use an object data mining approach to activity discovery by relying on the relevance weights of objects as the basis of activity discrimination rather than on sequence information.

Another example in this category is the work done by McCowan et al. [60] for automatic meeting analysis based on modelling interactions between individuals. Thus, a two-layer framework has been proposed to recognize individual and group actions in meeting. In the first layer, actions of individual participants (e.g. ‘‘writing’’, ‘‘speaking’’) are first measured using a variety of audiovisual features (such as speech activity, pitch, speaking rate, etc.).

These multimodal feature sequences are fused in the second layer to recognize actions belonging to the group as a whole. The result of this layer is group situations like “discussion’’, ‘‘note-taking’’, or ‘‘presentation’’.

The learning-based approaches have at their core a probabilistic reasoning method to, in the first instance, learn behaviour patterns and follow this to recognise activities or situations. A potential drawback of such approaches is the fact that learning behavioural patterns requires large amounts of activity historical data which can be difficult and costly to acquire.

3.3.3 Hybrid Approaches

In order to reduce the reliance on training data, several works try to incorporate domain knowledge into their approaches (e.g. [15][128][129][130]). For example, the activity classifier used in [130] is called the *situation lattice*, which is a mathematical model that is used to abstract and combine sensor data in a lattice structure. Through a learning process, the situation lattice can build the correlation between the abstracted sensor data and the high-level situations or human activities. It supports the representation and use of domain knowledge to incorporate semantic relationships between abstracting sensor data to tune the lattice as well as to incorporate temporal features in the inference process.

Brdiczka et al. [16] propose a two step situation learning framework. An initial simplified situation model is learned from a stream of perceptual events coming from different sensors in the environment by applying an automatic segmentation process with minimal human intervention. The human expertise is used only for providing the situation labels. This model is subsequently adapted to different users’ preferences by a supervised learning algorithm using feedback from users. That is, general situations, such as “Bob sitting on couch”, must be refined to obtain sub-situations incorporating the preferred system services in each sub-situation. Therefore, rather than preprogram the appropriate behaviours for a context-aware service, in this approach services adapt behaviour to individual preferences through feedback from the user. However, modelling the situation does not consider the history of states the user experiences i.e. it is a snapshot of the sensor reading at one point of time.

3.4 Service Adaptation Approaches

Many different solutions have been proposed by researchers to the problem of context-aware adaptation during service development and provision. The service adaptations can be classified according to whether they are performed at design-time or run-time; either at the service definition level or service instance level.

AdaptiveBPEL [131] is a service composition framework which aims at supporting the development of adaptive Web services compositions. This is achieved by leveraging the concept of aspects (originally from Aspect Oriented Software Development) to combine concerns (such as QoS) which are separately specified in BPEL processes and aspects. The adaptation process is driven by aspects weaving constructs generated based on a collaboration policy negotiated at runtime (by a built-in policy mediator) between the interacting endpoints.

To achieve process adaptation, a run-time aspect weaving middleware is integrated on top of a BPEL engine. The approach addresses the adaptation from the perspective of middleware. At runtime it transparently enforces QoS policies and dynamically adapts the composition instance through the ability to weave predefined extensions (such as encrypt outgoing messages) as Web service calls before, after or instead of an activity instance. However, the approach needs extensions to the existing Web service composition platforms, such as ActiveBPEL.

AO4BPEL [132], is an aspect-oriented extension to BPEL. In AO4BPEL, the main concern in workflows is the business logic, while crosscutting concerns (such as data validation and security) are specified using workflow aspects which provide better modularity and dynamics. Similar to AdaptiveBPEL [131], AO4BPEL proposes to solve the modularity problems using the aspect-oriented concepts in the context of workflow languages. However, there is a need to modify the BPEL engine to support aspects before and after executing each activity. In addition, since service logic is split up over many different files (aspects), this could make debugging a faulty service a difficult task [20].

eFlow [133] is a platform developed for specifying, enacting, and monitoring composite e-services (i.e., electronic services for e-business). Composite e-services are modelled (using graphs) as business processes, enacted by a service process engine. eFlow provides dynamic process change feature and distinguishes between *ad-hoc changes* (which apply to a single process instance) and *bulk changes* (which apply to many or all process instances). To achieve this adaptability, eFlow uses several constructs such as dynamic service discovery (i.e. service selection and binding), multiservice nodes (i.e. parallel execution of multiple equivalent services) and the notion of generic service that can be replaced with a specific set of services at process instantiation time or at runtime.

The eFlow’s migration manager allows users to modify running process instance(s) by migrating them from a source schema to a destination schema and without violating a predefined set of behavioural consistency rules. However, as eFlow uses its own process definition language and execution engine it remains vendor specific. In addition it tackles the adaptation on the code level. Moreover, the services composition should be adaptive not only to the events but also to the other adaptation triggers such as the change in business rules.

TRAP/BPEL [134] is a framework that adds autonomic behaviour to an existing BPEL service. The aim is to make an aggregate Web service continue its function even after one or more of its constituent Web services have failed. To achieve this aim, the TRAP/BPEL framework has been developed for automatically adapting BPEL services by monitoring the invocation of their partner Web services at runtime.

In detail, the framework monitors events such as faults and timeouts from within the adapted service which is augmented with a generic proxy that replaces failed services with predefined or newly discovered alternatives. TRAP/BPEL treats the adaptation of Web services compositions implicitly and achieves it only in the level of implementation at runtime. It extends neither the BPEL language nor its engine; however the realization of proxy causes extra versions of BPEL services. Moreover, addressing the adaptation both at design-time and at runtime is also needed.

Similar to TRAP/BPEL, **wsBus** [135] is a kind of broker which improves QoS by selecting appropriate services for execution at runtime. It is lightweight service-oriented middleware which is developed to address QoS concern of Web service compositions using broker pattern. The objective is to implement a customized messaging middleware optimized for the unique characteristics of SOAP. The wsBus introduces the concept of a virtual endpoint where a policy could be attached. During the service enactment, a handler bound to the virtual endpoint intercepts request and response messages and redirects messages to real services. The selection of services is based on monitoring data or QoS metrics. In this way, wsBus separates functional requirements (business logic) from non-functional requirements (such as QoS). However, since a large number of messages may be routed through it, the wsBus may become a bottleneck. In addition, wsBus focuses on runtime Web service composition instances adaptation and does not consider the adaptation at the service specification layer.

In the context of SaaS (Software as a Service), Ralph and Frank [82] present an approach that allows the generation of customization process out of variability descriptors that defines variability points for the process layer and related artefacts of process-based SaaS applications. SaaS model allows the provider to exploit economies of scale by hosting and providing the same application for several different customers; each of them has different requirements for the same basic application. This is achieved by providing an application template where some parts of the application remain unspecified (called variation points) or are defaulted and can be customized by each customer according to their needs. First, the customer needs to specify concrete values for the variability points of the application template. Depending on these values, different values might be permitted for subsequent variability points. Therefore variability points could be dependent on each other. The result of specifying variation points is an application solution which is then deployed at the SaaS provider hosting.

Further, a WS-BPEL process model that can then be used to guide a customer through the customization of the SaaS application is generated from the variability descriptors. However, using the variation points may not allow viewing the service variant in terms of the features that determine the difference between these variants in each usage context.

Another interesting work is the **Provop** approach [136], which provides a flexible solution for managing process variants following an operational approach to configure the process variant out of a basic process. This is achieved by applying a set of well-defined change operations (adaptations) to a common master (basic) process. Provop supports change patterns: Insert/Delete/Move process fragment and Modify process element attribute. Thereby, contextual information is utilized for enabling (semi)automated variant configuration. The framework has been implemented as an extension of the ARIS Business Architect (an existing BPM tool) in order to better cope with the high variability of business process models.

Choi et al. [137] propose an adaptation approach in a pervasive environment to support the modification of workflow at runtime. Each service is modelled as a sub workflow which can be inserted into the main workflow. If the context conditions are satisfied, that service will be executed. The adaptation takes place at the workflow definition level and is reflected in the running instance. However, their approach may not be sufficient to derive workflow variant; that is because this may involve rolling back executed tasks or adding new activities. They consider only the activities to be executed but not the activities that have already been executed.

Muller et al. [138] propose **AgentWork**, an interesting approach for workflow adaptation to customize the hospital cancer treatment workflow to suit each patient’s medical profile by adding and deleting tasks in the running workflow instance according to the predefined ECA (Event/Condition/Action) rules. The adaptation in this approach provides dynamic and automatic workflow adaptations and suggests and implements a reactive and predictive adaptation strategy. Thus, AgentWork uses temporal estimates to determine which remaining parts of a running workflow are affected by failures that may occur during workflow execution and is able to perform suitable adaptations in advance (predicatively). Reactive adaptation is performed when predictive adaptation is not possible; that is the adaptation is performed when the affected workflow part is to be executed.

AgentWork address the adaptation on the instance level and thus it may not be suitable to address the permanent changes that are due to business rules and which should be treated on the workflow definition level. In addition the adaptation mechanism cannot be applied to workflows developed without adaptability in mind, defined in standard languages (e.g. BPEL), or running in common engines (e.g. ActiveBPEL).

VxBPEL [20] is an adaptation language that is able to capture variability in processes developed in the BPEL language. VxBPEL provides the possibility to capture variation points, variants and realization relations between these variation points. Defining this variability information facilitates capture of a family of processes within one process definition and switching between these family members at run-time. VxBPEL works on the BPEL code level and the variants are mixed with the process business logic which may add complexity to the process developer task. Further, VxBPEL approach has been implemented in ActiveBPEL. In order to allow ActiveBPEL to execute VxBPEL, the engine must be adapted to recognize and store the new variability elements; in addition, a definition of behaviour during execution needs to be defined for these elements. Thus, the approach needs extensions for other BPEL engines.

Summary

Look at the approaches described above, the survey reveals that although the research in the area of context-aware adaptation is well established, there are still the following points missing that should be addressed:

1- The context management and adaptation logic in many existing approaches are handled at the code level by enriching the core logic of the service with code fragments responsible for context manipulation or adaptation rules. Significant examples of such approaches are Context Oriented Programming [139], AO4BPEL [132], and VxBPEL [20] which incorporates the variation points and variants inline in the service definition itself (i.e. BPEL code). However, as the service engineering process passes through the different phases (from analysis and design to the actual code development) the context and adaptation should be considered also in these phases.

2- Service modelling must be flexible enough to deal with constant changes – both at the business level and the technical level. The flexibility could be provided or addressed by incorporating variabilities into a system (e.g. [20][82][137]). Most of the approaches tackle service adaptation on the service instance or definition level by explicitly specifying some form of variation points. The problem is that, for example, each task in the service is modelled as a variation point, each ruled by its own clause to determine its inclusion or exclusion. This may be in contradiction with how the developer logically views the service variant i.e. in terms of the features that determine the difference between service variants in each usage context. Moreover, managing and understanding the service variants becomes more difficult when the number of variabilities and their relationships increase. Therefore, there is a need to capture the variability from a more logical point of view.

3.5 Conclusions

To summarise, as aforementioned there are still problems associated with current approaches for developing context-aware adaptive services. To correct these problems, a successful approach for context-aware adaptive service needs to meet the following requirements:

Requirement 1: Support for context variability

Several middleware and ontology-based models for describing context information have been developed in order to support context-aware applications. However, the context variability, which refers to the possibility to interpret the available context information from different perspectives to serve different applications, has been neglected in the existing context modelling approaches.

Requirement 2: Support for cross-domain context management infrastructure

The distribution of context information among different domains calls for a context management infrastructure (middleware) able to store, retrieve, and disseminate context information across domains. In addition, this middleware should provide developers with mechanisms to define their queries about context information of interest which may span different domains.

Requirement 3: Support for cross-domain situation recognition

A contextual situation recognition generic solution should recognize high-level situations based on the recorded (inferred) activities originated from different domains the user visits. In addition, this solution should also take into consideration the temporal relationships between the sequence of activities, and the fact that a situation can be characterized by a set of distinct activities but with no particular sequence.

Requirement 4: Support for service development and evolution

The context-aware service development usually involves several stages (e.g. analysis and design) prior to the actual code development. In addition, the service evolves according to the changes in the business rules and requirements. Therefore, the context and adaptation should be considered through the service development and evolution phases.

Requirement 5: The adaptation and business logics should be separated

Although the structure and behaviour of the service can be adapted to contextual information, the overall goal of the service core logic is indifferent to context change. Therefore, the adaptation to different contexts can be considered as an orthogonal task with respect to the core service logic. The separation of concerns is a promising approach in the design of such context-aware adaptive services where the core logic is designed and implemented separately from the context handling and adaptation logics.

Requirement 6: Providing the developer with constructs that facilitate capturing the service variants

There is a need to provide developers with mechanisms to capture the service variability from a logical point of view in order to easily manage and understand the service variants in each usage context.

Summary

To summarise, based on the investigation of the current techniques on context modelling and abstractions, context management middleware, and service adaptations, a new service engineering approach is required to eliminate the problems associated with these techniques, and therefore achieve universal context management and access control, and service adaptation with high automation.

The next chapter gives an overview of the proposed approach and its parts. It identifies the relevant areas of research and the corresponding contributions.

Chapter 4 Overview on the Proposed Approach

This thesis attempts to develop a new approach and related mechanisms to address the research question of how to achieve an effective and automated context-awareness in software services. In this respect, this thesis presents a conceptual model for developing context-aware adaptive services and software infrastructure for efficient context management that together facilitate the design and implementation tasks associated with such context-aware services. Therefore, two main areas of contribution are identified in this thesis: the context modelling, abstraction and management contribution, and the contribution to service adaptation (see Figure 4.1).

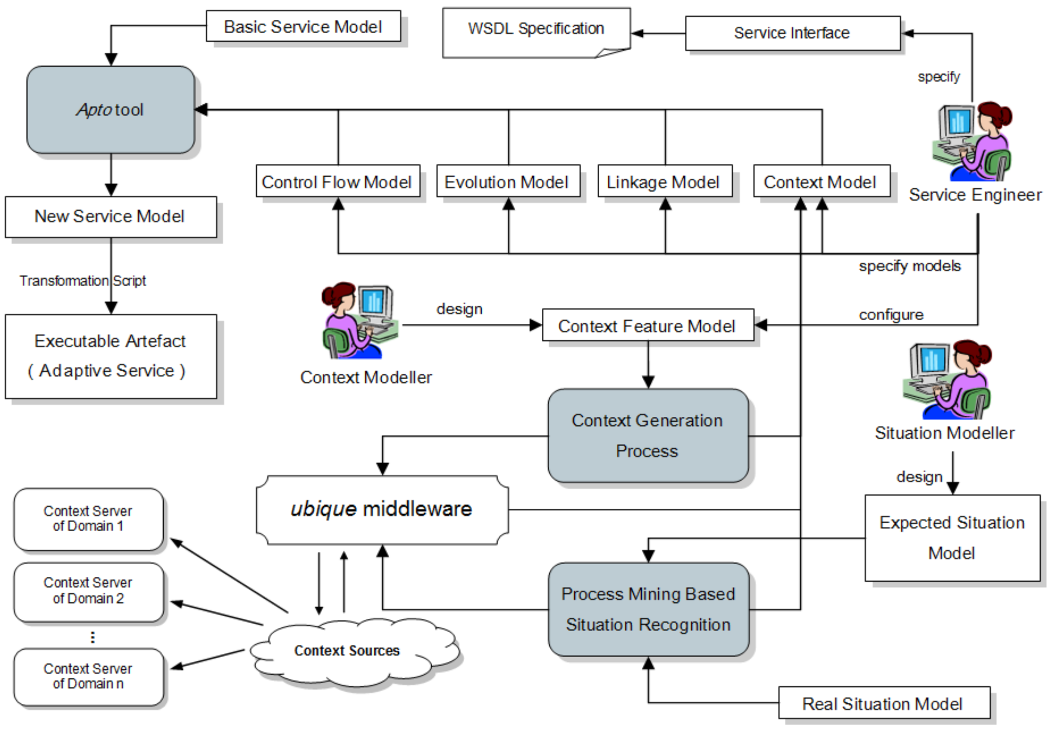


Figure 4.1 Overview on the proposed approach

4.1 Context Modelling, Abstraction and Management

To ease the development of pervasive applications it is necessary to provide universal context models and mechanisms to manage context. Thus, generic context models that can be reused by different applications and ease context sharing between systems are of interest. Therefore, a flexible product line based context model is introduced. It reduces application complexity and significantly enhances reusability of context information by providing context variability constructs (i.e. context features) to satisfy different application needs. This is achieved by devising context-specific features that can be shared among all applications as will be seen in Chapter 5.

On the other hand, the context information is naturally distributed among several domains. Therefore, the design of distributed storage, retrieval, and propagation mechanisms of context information between different domains becomes vital. Thus, to addresses the requirements of scalable context managements, Chapter 6 introduces *ubique,* a domain-based context management approach which allows developers to define domain-based context queries. In addition, it forms a robust context management infrastructure which enforces user's privacy and ensures efficient context information dissemination between domains.

Furthermore, the application adaptation is usually triggered by a change in context information i.e. a certain contextual situation. In order to effectively recognize contextual situations (which will drive the application adaptation) Chapter 7 focuses on the potential use of process mining techniques for cross-domain situation recognition. For this objective, the proposed situation recognition approach takes advantage of the above context management infrastructure.

4.2 Contribution to Service Adaptation

In pervasive environments the ultimate objective is to amplify human activities and demanding minimal attention from the user. Context-aware services aims to meet these objectives or requirements by adapting to a subset of the current context considered relevant to the task at hand such as the user location, time, and user situation. To this end, service modelling must be flexible enough to deal with constant changes – both at the business level (e.g. evolving business rules) and the technical level (e.g. contextual information and platform upgrades). The flexibility could be provided or addressed by incorporating variabilities into a system. Chapter 8 introduces two notions to capture the service variability in a logical and intuitive way: the *evolution fragment* and *evolution primitive*. Furthermore, the proposed mechanism could apply an adaptation to services modelled or developed without any adaptation possibility in mind and independently of specific usage contexts.

The next chapter is an attempt to address the main aforementioned limitations in the context modelling approaches by leveraging ideas from the SPL domain.

Chapter 5 Generative Feature-Based Context Model

As part of the proposed service engineering approach this chapter presents a flexible product line based context model which significantly enhances the reusability of context information by providing context variability constructs (i.e. context features) to satisfy different application needs. On one hand, commonality and variability management techniques from the SPL approach can be applied to handle context variabilities for serving different applications' needs. On the other hand, based on the context feature model, specific context (i.e. member of a product line) can be dynamically constructed by composing a specified set of context features.

5.1 Introduction

As aforementioned, different context knowledge could be extracted from the context repository by focusing on different views of the context information. For example, in the smart meeting room, a seat may be equipped with light and temperature sensors to reason about its occupation. The seat could be either free or occupied. Two occupation variants may be identified: occupied by an object or occupied by a person. These variants represent two facets to the same fact. Another example of context variability is the context information classification. For instance, the room temperature could be classified as low, moderate and high according to some specified temperature ranges; but these ranges could be different if the room type is a sitting or a sauna room. Therefore, in order for the middleware to serve different types of applications, SPL could be leveraged to provide context-specific programming abstraction or constructs that model the context variability.

This chapter focuses on dealing with context variability from the application requirement perspective. The proposed approach does not model the context information itself by using feature models as the feature models are less powerful than ontologies, and are more appropriate for expressing a subset of what ontologies can express [140]. Instead, the aim is to represent the context information from the requirement perspective via the feature model, the context primitives and their associations.

5.2 The Rationale of the Proposed Approach

The rationale behind this approach is as follows:

Firstly, in terms of modelling philosophy, in ontology modelling a concept is described by adding its details and implicitly defining in a bottom-up fashion the scope of the concept through the details. Whereas, in feature modelling, a concept is described by first setting its scope and hierarchically adding its details in a top-down fashion [140]. This approach is quite interesting as it allows the context modeller to devise, in a top-down fashion, generic and reusable context features which can be shared among all applications that need to use the available context information. The relationships between context features express the context variability from the application point of view.

Secondly, according to the proposed working definition of the context illustrated in Figure 5.1, the context knowledge is composed of a set of small contextual knowledge pieces namely *context primitives* which include context entities, attributes, associations, and rules. Each context feature corresponds to a specific set of context primitives. The focus is a concept representing the point of view the application is interested in looking at the current context. Each focus corresponds to a specific set of context features. Given a focus, a relevant subset of these pieces will be used to generate per-application customized contextual knowledge. Obviously, considering only the relevant context primitives would improve the reasoning performance and reduce response time which is a vital issue in the pervasive environment.

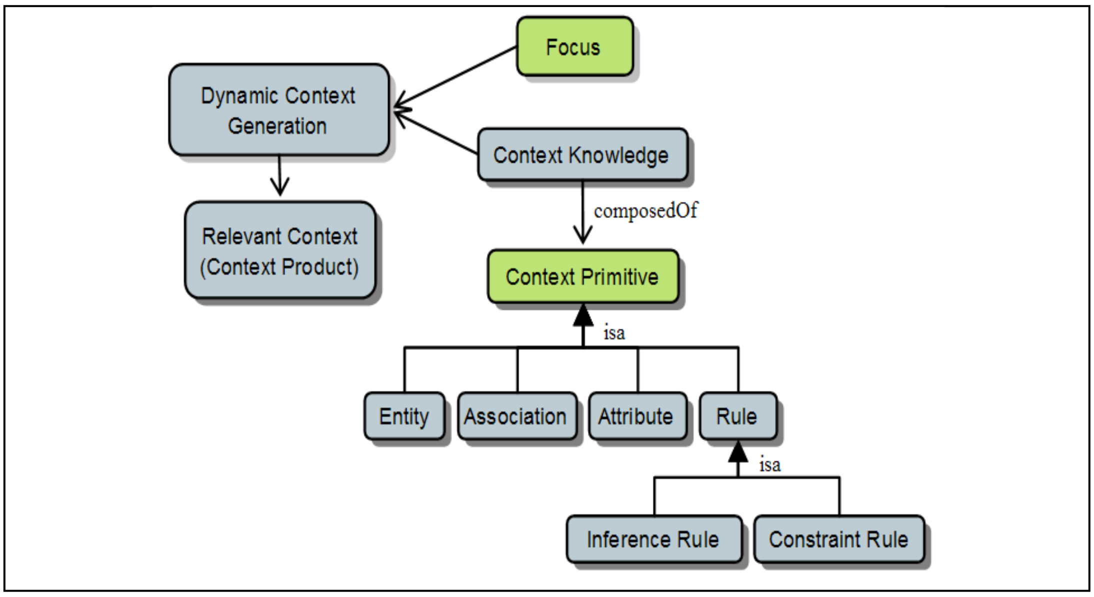


Figure 5.1 The working definition of the context

Thirdly, applications use context queries to retrieve the set of context information that adhere to some conditions. Some context queries are difficult to be defined using general-purpose querying mechanisms (e.g. SPARQL [1]). In addition, the application developer may not have enough knowledge about context semantics, in order to describe queries correctly.

Finally, as developers usually do not have full understanding of the context internal semantic, "promoting" the context information using the feature model will enable the contextual knowledge visibility from different views in a top-down fashion. Another advantage is that these context features might be shared between applications which significantly enhances the reusability of context information and reduces application complexity.

[1] http://www.w3.org/TR/rdf-sparql-query/

5.3 The Conceptual Model for Context Management

The concepts of features have been imported from Feature Oriented Domain Analysis (FODA) [94]. In FODA, features are essential abstractions that both context consumer and provider understand. Thus, the main concept in the feature description language FODA is the feature itself. Here a feature is a set of context primitives that is relevant to some stakeholder from a specific "focus" point of view. Figure 5.2 depicts the proposed conceptual metamodel. The concepts of the conceptual metamodel were identified and grouped into two different sections: the context related concepts (white), and the context features concepts (shaded).

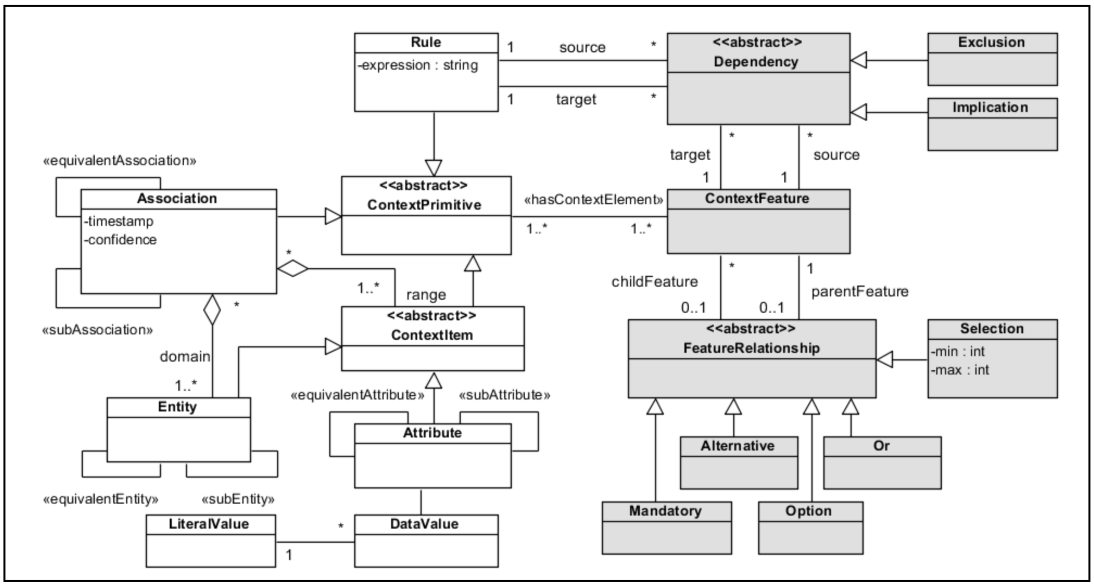


Figure 5.2 The Conceptual Meta-Model

The main construct for representing contextual knowledge is the *ContextPrimitive* which represents the base context constructs (primitives) mentioned above: entity classes, entity attributes, entity associations, and rules.

• Entity class: represents a group of entities (e.g. users, places, devices, etc.) sharing some properties.

• Attribute class: represents an entity's attributes e.g. preference, position, temperature, etc.

• Association class: represents a relationship between one entity and either another entity or an attribute.

Further optional modelling constructs are additional facts about the entities and attributes. These are: *specialization* and *equivalence* relationships that may be specified between two entity classes, two attribute classes, or two association classes.

Two types of rules could be identified: (i) *Consistency* rules provide a mechanism for context consistency by specifying conditions that must be held in the context information. For example, a consistency rule could specify that if the person is cooking, they must be in the kitchen. (ii) *Inference* rules are used to generate new context information after reasoning on the existing one. For example, an inference rule could conclude that a person is sleeping if she is in the bed room, the light is off and it is night-time.

5.4 Context as a Dynamic Product Line

As already mentioned the context evolves dynamically according to the focus and that context is a set of contextual elements that are assembled and instantiated according to the focus. This section explains how the context management middleware can dynamically generate the per-application context information given a set of features.

In fact, both middleware and context models are strongly interdependent since the complexity of a context model determines the complexity of context management by a middleware. Coutaz et al [53] presents this relationship as a conceptual framework that interconnects an ontological foundation for context modelling with the middleware (runtime infrastructure).

5.4.1 Feature-based Context Modelling

In order to identify what of the context information is eligible for being modelled as a feature, simplified criteria have been adopted which are composed of the three steps shown below, followed by the correspondent modelling decisions:

1- Identify the context information required by the application adaptation (e.g. user location). This should be represented by a generic feature in the feature model.

2- Identify the different interpretations of the currently available context information in order to be shared by all application instances (e.g. room-, floor-, and building-resolution user location information). These interpretations should be represented by different feature variants.

3- Regrouping the different identified context features into a logical hierarchy of features in a top-down manner that could be used by different applications. The result is a *context feature model*.

The context feature model should be published in a public registry. When an application developer needs to use context information, they read the XML files representing the different context features from different perspectives. The developer is able to understand the context semantics; then they are able to configure the feature model and use the middleware services to get the necessary context information.

Although a feature model can represent context commonalities and variabilities in a concise taxonomic form, features in a feature model are merely symbols. Mapping features to the context ontology gives them semantics. In the following section the proposed approach to mapping the feature model to the ontology context model is described.

5.4.2 Annotated Context Model

An overview of the proposed approach is shown in Figure 5.3. A context model family is represented by the context feature model and the ontology-based context model (OCM). The elements of OCM namely the context primitives may be annotated using **Existence Conditions** (ECs) and **Meta-Statements** (MSs). These annotations are defined in terms of features and feature attributes from the feature model, and can be evaluated with respect to a feature configuration. An EC attached to a context primitive indicates whether the primitive should exist in or should be removed from a context product. MS is mainly used to modify or compute the attributes of context model element. This is important for managing context variants as we will see in the case study in Section 5.6. For example, evaluating the following MS boils down to evaluating its expression which results in modifying the property value `minimumJournalRank` of the `FMConfiguration` entity from 100 to the value of the variable `$minimumJournalRankVariable`.

```
<metastatement name="MS1">
  <expression>
     PREFIX cxt:&lt;http://www.napier.ac.uk/candel#&gt;
     PREFIX xsd:&lt;http://www.w3.org/2001/XMLSchema#&gt;
     DELETE
       { cxt:FMConfiguration cxt:minimumJournalRank "100.0"^^xsd:float }
     INSERT
       { cxt:FMConfiguration cxt:minimumJournalRank
"$minimumJournalRankVariable" ^^xsd:float }
   </expression>
</metastatement>
```

The value of the variable `$minimumJournalRankVariable` can be calculated by evaluating the following meta-statement variable expression which refers to the attribute `minimumJournalRank` of the feature `HavingJournalPublications` in the context feature model.

```
<metastatementVariable name="minimumJournalRankVariable"
expression="//feature[@id='HavingJournalPublications']/minimumJournalRank">
</metastatementVariable>
```

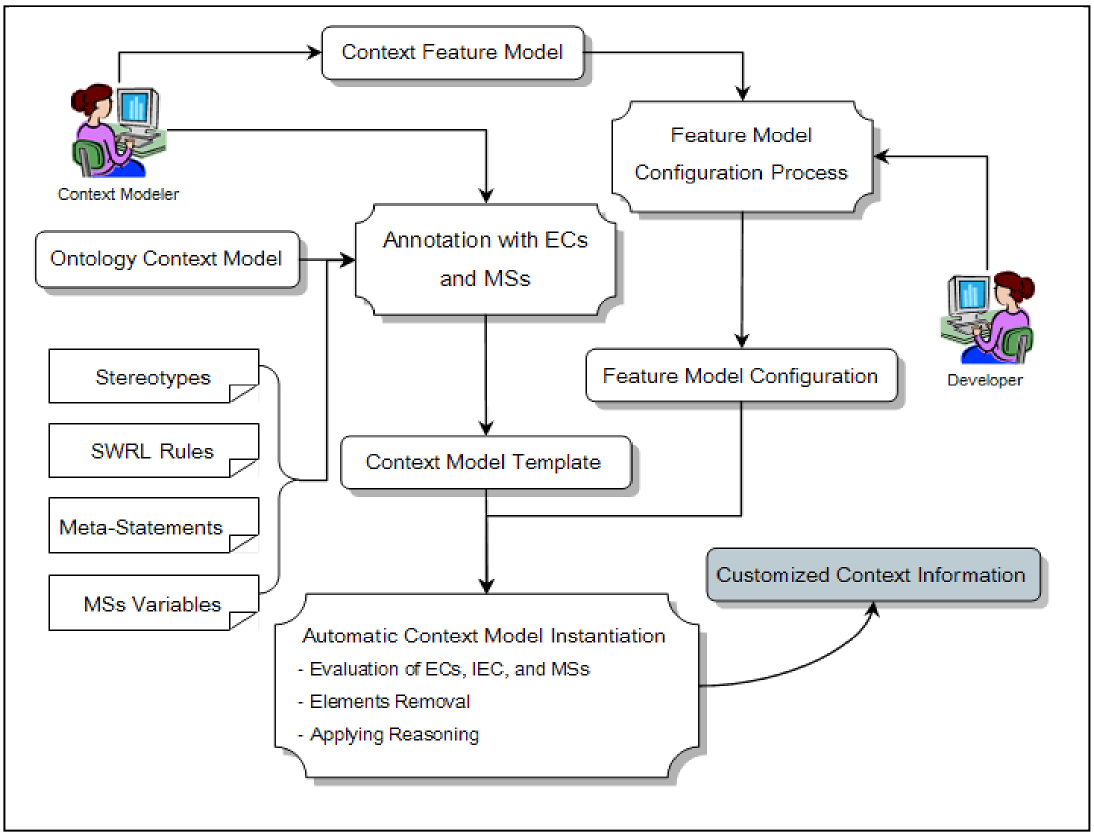

Context Feature Model
Context Modeler
Feature Model Configuration Process
Ontology Context Model
Annotation with ECs and MSs
Developer
Stereotypes
SWRL Rules
Meta-Statements
MSs Variables
Feature Model Configuration
Context Model Template
Customized Context Information
Automatic Context Model Instantiation
- Evaluation of ECs, IEC, and MSs
- Elements Removal
- Applying Reasoning

Figure 5.3 Overview of the proposed approach

An instance of a context model family, which we call context product (CP), can be specified by creating a feature configuration based on the context feature model. Based on the context feature model configuration, the corresponding context product is generated automatically. The generation process, which is model-to-model transformation, involves evaluating the ECs and MSs with respect to the feature configuration, removing the context primitives whose ECs evaluate to false and, possibly doing additional processing such as removing related context primitives.

Obviously, a particularly interesting form of ECs is a Boolean expression over a set of variables each of which corresponds to a feature from the feature model. Given a feature configuration, the value of a feature variable is true if and only if the corresponding feature is included in the feature configuration. In the prototype implementation two forms of expressions are used: (i) Boolean expressions in Disjunctive Normal Form (DNF), or (ii) more general XPath expressions which can access feature attributes and use other XPath operations, as long as the XPath expression evaluates to a Boolean value. The EC is represented by one or more **stereotypes**. For example, the stereotype «!f1&&f2||f3» in DNF denotes the Boolean expression $\overline{f_1}.(f_2 + f_3)$. Once created, the stereotype is available for annotating context primitives.

On the other hand, the ECs should be interpreted with respect to the OCM containment hierarchy. In other words, if a context primitive container is removed all the contained context primitives are removed. For example, if entity X is a sub-entity of the entity Y, removing Y requires removing X as well.

5.4.3 Implicit Existence Condition (IEC)

Context primitives that are not explicitly annotated will have **Implicit Existence Condition** (IEC). The IEC for a context primitive can be provided based on the existence conditions of other context primitives and on the syntax and semantics of the OCM. For example, according to the ontology syntax, an Object Property requires a class at each of its ends. Thus, a reasonable choice of IEC for an object property would be the conjunction of the ECs of both classes. This way, removing any of the classes will also call for the removal of the object property. IECs reduce the necessary annotation effort of the developer.

Table 5.1 shows the choice of IECs for the context primitives. An IEC for a given primitive is assumed based on its type.

Table 5.1 IEC for different context primitives

Primitive Type	**Implicit Existence Condition**
Association	Conjunction of the EC of the two Entities associated with Association type.
SubEntity	The EC of the Parent Entity is evaluated to true.

SubAssociation	The EC of the Parent Association is evaluated to true.
Attribute	The EC of the Entity is evaluated to true.
Rule	True iff the ECs of all required rules are true and the ECs of all its excluded rules are false.

5.5 Context Information Generation

A context information generation process involves computing MSs and ECs, and removing elements whose ECs are false. The complete context product instantiation algorithm can be summarized as follows:

1- Evaluation of MSs and explicit ECs: The evaluation is done while traversing the OCM containment hierarchy in depth-first order. Children of context primitives whose ECs evaluate to false are not visited because they will be removed.

2- Removal Analysis: Removal analysis involves computing IECs. The IECs can be computed in a single additional pass after evaluating explicit ECs. In addition, in this step all the individuals and statements whose subjects are included in the elements to be removed are also marked to be removed. For example, if the *Room* entity is known to be removed, all its individuals and all triples whose subject is of type *Room* should be marked to be removed.

3- Primitive Removal: In this step, primitives whose ECs are false or which are marked to be removed are removed.

4- Applying Reasoning: In order to interpret the remaining context information from the perspective specified by the context feature configuration, it is necessary to apply the corresponding remaining rules. The result of the reasoner will be the context product.

Different rule-based systems provide different logical inference support for context reasoning. To reason about ontologies, Pellet[2] for example can be

[2] http://clarkparsia.com/pellet/

applied a description logic reasoner. The Semantic Web Rule Language (SWRL[3]) has been used on top of OWL for interpreting context using domain specific rules and producing new facts. In the implemented prototype the rules have been specified by using the SWRL and the Java Expert System Shell (Jess[4]) has been used as the inference engine. However, the approach could be extended to use other reasoner types.

5.6 Case Study: Conference Advisor Application

5.6.1 Objective

The objective of this case study is to illustrate and evaluate the proposed approach for product line based context modelling and the service adaptation. This case study first applies the approach of product line based context modelling to model the context information available in a conference venue. Furthermore, it shows how and how effective a customised version of a service application could be generated using the *Apto* approach.

For this objective the following scenario is considered: Alice is a researcher going to attend a conference in London. Once she has arrived at the conference building, she decides to contact expert researchers. The expertise of a researcher could be interpreted in different ways e.g., depending on her publications in journals, on her patents or awards, etc.

5.6.2 Illustration and Evaluation of Product Line based Context Model

The key feature of the proposed modelling approach is its ability to support variable ontology reasoning in a pervasive environment. In this case study some concepts from the SO4PC ontology [141] have been used for expressing context information associated with persons, time, and spaces.

[3] http://www.w3.org/Submission/SWRL/
[4] http://www.jessrules.com

Another ontology has been used for describing the research related concepts. Figure 5.4 shows a snippet of the classes and properties used in the ontology. The complete ontology used in this case study can be found in Appendix B.

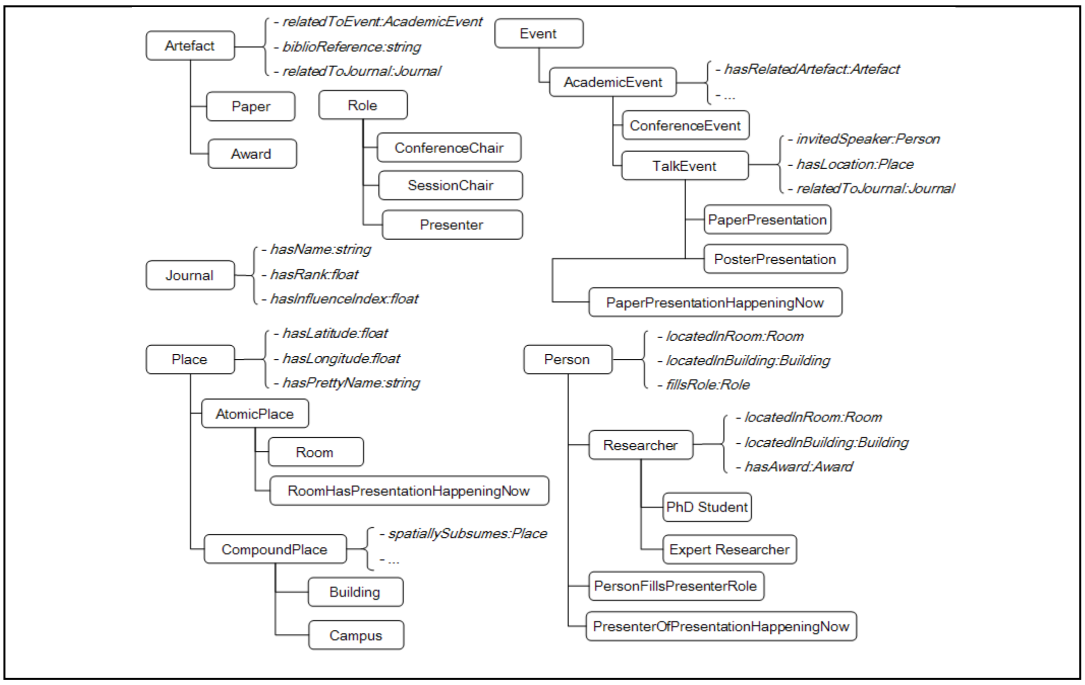
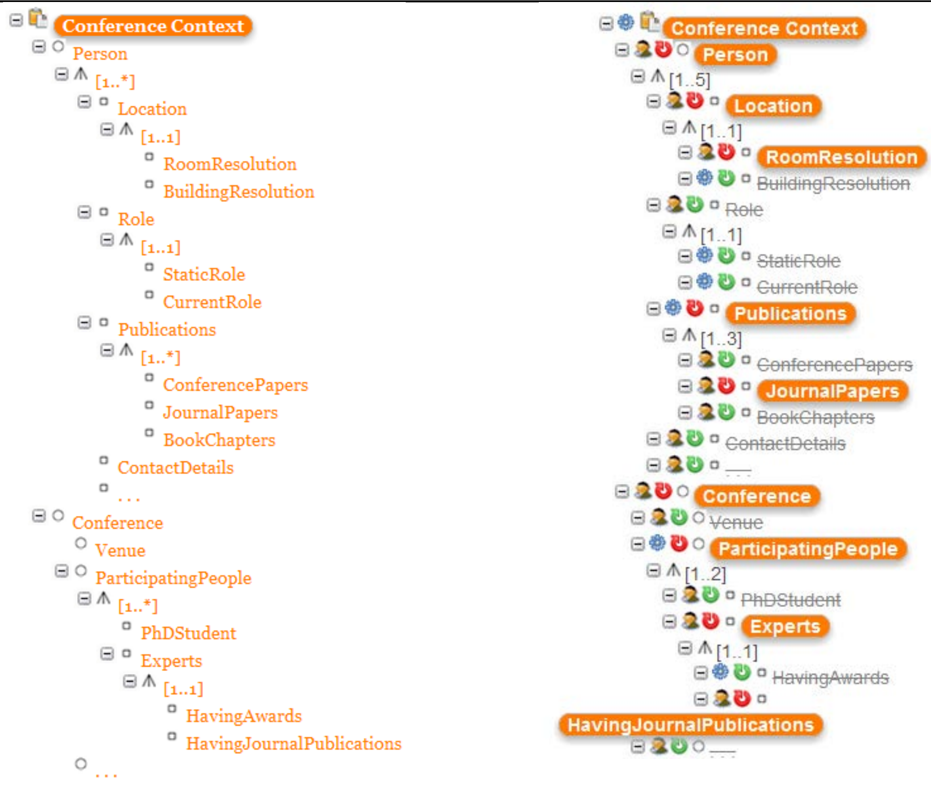


Figure 5.4 A snippet of the used ontology

Figure 5.5 (a) shows an example of a context feature model which represents different features that could be shared among different applications. For example, if the `Location` feature has been selected, then two mutually-exclusive options are available; either as a room resolution; or as a building resolution. In either case, different concepts, properties, attributes and rules should be considered. In a similar manner, the `Role` feature regroups two features: the static role (e.g. `Reviewer` and `OrganisingCommitteeMember`) or the current role played during the conference (e.g. `Presenter` and `SessionChair`). Figure 5.5 (b) shows one possible context feature configuration.



(a) Context Feature Model (b) Feature Model Configuration

Figure 5.5 Example of context feature model

Each feature may have several attributes. For example, in Figure 5.6 that shows a part of the feature model configuration XML file, the `HavingJournalPublications` feature has two attributes: `value` which indicates the selection of the feature or not, and `minimumJournalRank`. This feature allows the retrieval of researchers who have been published in journals whose rank is superior to the attribute `minimumJournalRank` value.

```
<configuration model="Context Feature Model">
  <feature id="Person">
     <value>1</value>
   </feature>
   <feature id="Location">
     <value>1</value>
   </feature>
   <feature id="RoomResolution">
     <value>1</value>
   </feature>
   <feature id="BuildingResolution">
     <value>0</value>
   </feature>
   <feature id="Experts">
     <value>1</value>
   </feature>
   <feature id="HavingAwards">
     <value>0</value>
   </feature>
   <feature id="HavingJournalPublications">
     <minimumJournalRank>350</minimumJournalRank>
     <value>1</value>
   </feature>
    ...
</configuration>
```

Figure 5.6 Feature model configuration

As aforementioned, in order to link the context feature model to the context primitives, stereotypes are used to annotate ontology elements as well as the SWRL rules. Figure 5.7 shows a snippet of the XML file containing the available stereotypes to be used for annotation. Each stereotype expression is expressed, as described above, in terms of the features' values of the context feature model.

```
<stereotypes>
<stereotype name="Person" expression="$ConferenceContext"></stereotype>
<stereotype name="RoomResolution" expression="$RoomResolution ||
$BuildingResolution"></stereotype>
<stereotype name="BuildingResolution" expression="$BuildingResolution"></stereotype>
<stereotype name="Paper" expression="$ConferencePapers || $JournalPapers ||
$Experts"></stereotype>
<stereotype name="ConferencePaper" expression="$ConferencePapers"></stereotype>
<stereotype name="Conference" expression="$Conference"></stereotype>
<stereotype name="JournalPaper" expression="$JournalPapers"></stereotype>
<stereotype name="StaticRole" expression="$StaticRole"></stereotype>
<stereotype name="CurrentRole" expression="$CurrentRole"></stereotype>
<stereotype name="Conference" expression="$Conference"></stereotype>
<stereotype name="Location" expression="$Location || $Venue"></stereotype>
<stereotype name="Publications" expression="$Experts || $Publications"></stereotype>
<stereotype name="Experts" expression="$Experts"></stereotype>
<stereotype name="ExpertHavingAwards" expression="$HavingAwards"></stereotype>
<stereotype name="ExpertHavingJournalPublications"
expression="$HavingJournalPublications"></stereotype>
...
</stereotypes>
```

Figure 5.7 Example of available stereotypes

Figure 5.8 shows a sample of the annotated ontology elements. The `Label` property is used to specify the correspondent stereotypes of each element.

```
<owl:Class rdf:ID="CompoundPlace">
  <rdfs:subClassOf rdf:resource="#Place"/>
  <rdfs:label>BuildingResolution</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="Building">
  <rdfs:subClassOf rdf:resource="#CompoundPlace"/>
  <rdfs:label>BuildingResolution</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="Room">
  <rdfs:subClassOf rdf:resource="#AtomicPlace"/>
  <rdfs:label>RoomResolution</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="MeetingRoom">
  <rdfs:subClassOf rdf:resource="#Room"/>
  <rdfs:label>RoomResolution</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="RoomHasPresentationHappeningNow">
  <rdfs:stereotype>CurrentRole</rdfs:stereotype>
  <rdfs:subClassOf rdf:resource="#Room"/>
  <rdfs:label>RoomResolution</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="Journal">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:label>ExpertHavingJournalPublications</rdfs:label>
</owl:Class>
...
<owl:ObjectProperty rdf:ID="relatedToJournal">
  <rdfs:domain rdf:resource="#Artefact"/>
  <rdfs:range rdf:resource="#Journal"/>
  <rdfs:label>ExpertHavingJournalPublications</rdfs:label>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasLocation">
  <rdfs:domain rdf:resource="#OrganisedEvent"/>
  <rdfs:range rdf:resource="#Place"/>
  <rdfs:label>Location</rdfs:label>
</owl:ObjectProperty>
...
```

Figure 5.8 Example of annotated ontology

On the other hand, as mentioned above, MSs can be expressed using XPath. As an example, the MS represented in Figure 5.9, uses the SPARQL Update[5] expression to update the datatype property `minimumJournalRank` of the entity `FMConfiguration` (see Appendix B) by a value retrieved from the variable `$minimumJournalRankVariable` whose value is determined by the XPath expression of the variable `minimumJournalRankVariable` in Figure 5.10. The result of applying this MS is to change the value of the

[5] http://www.w3.org/Submission/SPARQL-Update/

minimumJournalRank datatype of the entity FMConfiguration from 100.0 to 350 (as in the configured feature model of Figure 5.5).

```
<metastatements>
 <metastatement name="MS1">
  <expression>
     PREFIX cxt:&lt;http://www.napier.ac.uk/candel#&gt;
     PREFIX xsd:&lt;http://www.w3.org/2001/XMLSchema#&gt;
     DELETE
       { cxt:FMConfiguration cxt:minimumJournalRank "100.0"
                  ^^xsd:float }
     INSERT
       { cxt:FMConfiguration cxt:minimumJournalRank
                 "$minimumJournalRankVariable" ^^xsd:float }
   </expression>
   <stereotype>ExpertHavingJournalPublications</stereotype>
 </metastatement>
...
<metastatements>
```

Figure 5.9 Example of meta-statement

```
<metastatementsVariables>
  <metastatementVariable name="minimumJournalRankVariable"
expression="//feature[@id='HavingJournalPublications']/minimumJournalRank">
  </metastatementVariable>
...
</metastatementsVariables>
```

Figure 5.10 Example of meta-statement variable

Figure 5.11 shows a sample set of the annotated SWRL rules. For example, Rule1 is used to reason about the paper presentations that are currently taking place. To determine if the researcher is an expert we have two options: by choosing the HavingAwards or HavingJournalPublications features. The Rule4 corresponds to the former option. The Rule2 and Rule3 correspond to the latter option and are used to determine if the researcher has been published in journals having a specified minimum rank and minimum influence index respectively. Rule5, Rule6 and Rule7 are among the rules used to reason about the person location in building resolution. The stereotype of the rule is specified by the *stereotype* element.

```
<swrlrules>
 <swrlrule name="Rule1">
    <expression> PaperPresentation(?p) ^ hasStartDateTime(?p, ?s) ^
hasEndDateTime(?p, ?e) ^ swrlb:currentDateTime(?c) ^ swrlb:beforeTime(?s, ?c) ^
swrlb:beforeTime(?c, ?e) -> PaperPresentationHappeningNow(?p) </expression>
    <stereotype>CurrentRole</stereotype>
 </swrlrule>
 <swrlrule name="Rule2">
   <expression>Researcher(?r) ^ authorOf(?r, ?p) ^ relatedToJournal(?p, ?j) ^
hasRank(?j, ?rank) ^ FMConf(?conf) ^ minimumJournalRank(?conf, ?minRank) ^
swrlb:greaterThan(?rank, ?minRank) -> ExpertResearcher(?r)
   </expression>
   <stereotype>ExpertHavingJournalPublications</stereotype>
 </swrlrule>
 <swrlrule name="Rule3">
   <expression>Researcher(?r) ^ authorOf(?r, ?p) ^ relatedToJournal(?p, ?j) ^
hasInfluenceIndex(?j, ?II) ^ FMConf(?conf) ^ minimumInfluenceIndex(?conf, ?minII) ^
swrlb:greaterThan(?II, ?minII) -> ExpertResearcher(?r)</expression>
   <stereotype>ExpertHavingJournalPublications</stereotype>
 </swrlrule>
 <swrlrule name="Rule4">
   <expression>Researcher(?r) ^ authorOf(?r, ?p) ^ hasAward(?p, ?award) ^
FMConf(?conf) ^ topAwardName(?conf, ?award) -> ExpertResearcher(?r)</expression>
   <stereotype>ExpertHavingAwards</stereotype>
 </swrlrule>
 <swrlrule name="Rule5">
   <expression>AtomicPlace(?x) ^ CompoundPlace(?y) ^ isSpatiallySubsumedBy(?x, ?y) -
> spatiallySubsumes(?y, ?x)</expression>
   <stereotype>BuildingResolution</stereotype>
 </swrlrule>
 <swrlrule name="Rule6">
   <expression>AtomicPlace(?x) ^ CompoundPlace(?y) ^ CompoundPlace(?z) ^
isSpatiallySubsumedBy(?x, ?y) ^ isSpatiallySubsumedBy(?y, ?z) ->
isSpatiallySubsumedBy(?x, ?z)</expression>
   <stereotype>BuildingResolution</stereotype>
 </swrlrule>
 <swrlrule name="Rule7">
   <expression>Researcher(?r) ^ locatedInAtomicPlace(?r, ?p) ^
isSpatiallySubsumedBy(?p, ?cp) -> locatedInCompoundPlace(?r, ?cp)</expression>
   <stereotype>BuildingResolution</stereotype>
 </swrlrule>
...
</swrlrules>
```

Figure 5.11 Example of annotated SWRL rules

Figure 5.12 shows an example of the retrieved context information after sending the feature model configuration (of Figure 5.5(b)) to the implemented middleware prototype.

```
<ExpertResearcher rdf:ID="Alice">
    <rdf:type rdf:resource="#Researcher"/>
    <authorOf>
      <Paper rdf:ID="FirstPaper">
        <relatedToJournal>
          <Journal rdf:ID="JournalOne">
            <hasRank rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
>204.0</hasRank>
            <hasInfluenceIndex rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
>15.83</hasInfluenceIndex>
            <hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">JOURNAL
OF THE ACM</hasName>
          </Journal>
        </relatedToJournal>
        <biblioReference
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Product Line based Context
Modelling </biblioReference>
      </Paper>
    </authorOf>
    <authorOf rdf:resource="#SecondPaper"/>
    <locatedInRoom rdf:resource="#C33"/>
</ExpertResearcher>
...
```

Figure 5.12 The retrieved context information

5.6.3 Summary

In conclusion, this case study has shown that the context modelling approach and the related tool are capable of serving different applications' needs of context information. This is achieved by "promoting" the context information via a context feature model capable of expressing the context variability. Using this approach the context modeller is able to devise a generic context feature model which includes different context features corresponding to different interpretations of the same fact. For example, in this case study, the experience of a researcher has been interpreted according to two criteria corresponding to two features: `HavingAwards` and `HavingJournalPublications`. From the developer point of view it is an intuitive and easy method to specify and retrieve a version of the available context corresponding to their perspective. This way the developer is alleviated from the burden of reasoning about the available context and specifying context queries. In this case, they need to configure the context feature model by specifying the features they are interested in and their attributes. However, this comes with a price. In order to serve different applications the context modeller has to have a clear understanding of the context semantics and devise a generic context feature model that covers all possible interpretations of the different facts. This includes devising one or more feature models each of which focuses on a different topic. For example, the case study shows one feature model focusing on the conference context; another feature model could be devised to focus on the researcher himself. In addition, the context modeller has to annotate the ontology with the devised stereotypes which may not be an easy task when the ontology is huge. However, the concept of the implicit existence condition (IEC) alleviates the modeller from annotating every context primitive. The result is facilitating the developer task and obviously by considering only the context primitives corresponding to the specified context features the reasoning performance would be improved.

5.7 Conclusion

This chapter has presented an approach for supporting the development of context-aware services based on a flexible product line based context model. The proposed approach to model the context information allows the context modeller to specify the context information in a high-level and logical way that regroups context variabilities; and provides service developers with mechanisms (context features configuration) to express their needs from context information. The novelty of this approach lies in (i) the introduction of the context variability concept, (ii) a hybrid context modelling approach which takes advantage of the ontology-based modelling approaches and at the same time map the available contextual knowledge to a set of context features that could be shared and reused by different applications, and (iii) a generative approach to provide applications with the context information they need according to the chosen context features. The result is a more intuitive way to represent context and improve overall systems performance.

However, this approach can be applied to “promote” the context information available in one administrative (spatial) domain. In the next chapter the focus will be on proposing a context management middleware architecture which allows developers to maintain context queries spanning different domains. In this respect, a collaboration protocol between context servers available in different domains for context storage, retrieval, and dissemination is thus proposed.

Chapter 6 *ubique*: Cross-Domain Efficient and Privacy-Ensuring Context Management Middleware

In pervasive environments, context-aware services require a global knowledge of the context information distributed in different spatial domains in order to establish context-based interactions. Therefore, the design of distributed storage, retrieval, and dissemination mechanisms of context information across domains becomes vital. In such environments, there is a need for the collaboration between different context servers distributed in different domains; thus, the need for generic APIs and an appropriate communication protocol allowing context information exchange between different entities: context servers, context providers, and context consumers. As a solution this chapter proposes *ubique*, a distributed middleware for context-aware computing that allows applications to maintain domain-based context interests to access context information about users, places, events, and things - all made available by or brokered through the *home domain server*. This chapter proposes also a new cross-domain protocol which ensures the user's privacy and the efficiency of context information dissemination. *ubique* has been robustly built upon the Jabber protocol which is widely adopted open protocol for instant messaging and is designed for near real-time communication. Simulation and experimentation results show that *ubique* well supports robust cross-domain context management and collaboration.

6.1 Introduction

Context-awareness is the cornerstone to achieve the vision of pervasive computing. It refers to the capability of an application or service being aware of its physical environment or situation (e.g., context) to respond proactively and intelligently based on this awareness [142].

Context-awareness should be supported by a context management system that allows the automatic discovery, retrieval and exchange of context information distributed in different administrative (spatial) domains. Such a system must perform its functions in a pervasive computing environment that involves mobile users and devices. The proposed context management middleware is based on the notion of *context domain* explained in [29] which organizes the pervasive environment hierarchically and establishes the context management scope. A context domain is defined as an abstraction of a spatial area which has a clear boundary and it is built on top of the traditional notion of network domain. Essentially, context domain establishes (i) the place and responsibility of context instances storage; (ii) the responsibility for managing context providers and consumers inside the domain; and (iii) a set of sub-domains.

Although users are more interested in the context information related to their location, other context information from other domains may also be relevant to the current task at hand. For instance, a dynamic recalculation of the quickest routes for a trip involves acquiring the latest contextual information such as traffic congestion from remote sources. In this respect, we can imagine a domain-based context management system where the context information available in each domain is managed by a separate context server. While moving, the user roams across domains. In addition, each domain may maintain its own sensors and mechanisms for inferring context related to this user. Consequently, collaborative context management across domains is needed.

In particular, an efficient cross-domain context management middleware system for such a setting needs to fulfil key requirements that include (as mentioned in Section 3.2.1): (i) domains of context perception, (ii) uniform API interface for accessing context servers, (iii) efficient context information dissemination, (iv) support of cross-domain reasoning, (v) dynamic matching between context providers and consumers, and (vi) support for privacy.

Therefore, this chapter proposes *ubique*, a new domain-based context management infrastructure for context management and dissemination between context providers, context consumers and context servers, and a set of APIs for interfacing between these entities. *ubique* fulfils the above mentioned key requirements and it forms an underlying robust and generic infrastructure for context management, which significantly simplifies the development of context-aware pervasive applications.

6.2 Context Dissemination Problem

Consider a simple context dissemination scenario: a user is subscribed to a context server (CS) located in domain A; namely CSA. This server maintains the profile information of its subscribed users and maintains a sensor infrastructure for domain A. This server is called the *home domain server* (HDS) of its subscribed users. Likewise, the context server CSB maintains the users' profiles and physical context information of domain B. Obviously as long as the user is still in the domain A the scenario is rather simple; all the context information needed by the application about this user exists in CSA. However, when the user moves from A to B the context information related to the user maintained by CSA and CSB (such as location or environment context information) may become relevant to the applications interested in the user's context. In this case, the CSB is called the *visited domain server* (VDS). Thus, the applications have to be provided by mechanisms through which they can know the domains visited by the user at any point of time and the context information gathered about the user in these visited domains.

One possible solution is to use distributed tuple spaces (e.g., Confab [4]). Confab architecture structures context information into distributed tuple-spaces called *infospaces*, which store tuples about a given entity. An application interested in a certain context, builds a context query using the address of the responsible *infospace*. Although distributed *infospaces* contribute to decrease the context management overhead in a distributed environment, this distribution is not kept transparent to applications, which must know what infospace contains the desired context information. Another possible solution is to maintain in the HDSs "links" to the VDSs. In this case, in order to handle the application's queries about the users (or entities) over distributed domains, the HDS may have to distribute queries over the VDSs (e.g. [4][7]). However, this approach requires maintaining the link list of the VDSs, and may degrade the system performance as it requires distributing the application query over different servers and regrouping the result.

On the other hand, the notion of home and visited domains are also used by mobile telephone networks like GSM. The main idea used in these networks is that users have their "home domains" in which their context is gathered but when they roam to another domain this domain becomes a "visited domain". When a mobile device moves into a different domain, the server of the visited domain inter-links the mobile device and its home server. The home server redirects query statements to the server of the visited domain, which finally dispatches it to the mobile device. This is achieved by using the Home Location Register (HLR) and Visitor Location Register (VLR) approach of the GSM user profile database [143]. This approach addresses the location-awareness problem by minimizing the invocation of multiple updates in the home node each time a mobile user changes his/her location. However, the effectiveness of this mechanism is questionable for other types of context information, as it requires the application to submit their queries through a web of pointers from the home node to the visited node of the mobile user [144].

The main problem of context dissemination across domains originates from the observation that in a distributed system there is an obvious trade-off between costs of updates and costs of requests; i.e. between the communication cost introduced by the complete dissemination of the context data to the home node and the degree of dissemination that is eventually necessary. This has a direct impact on the achieved system performance and on the provided context precision. For example, when the volume of context data or the rate of change is high, providing high precision context value tends to degrade the performance; on the contrary, optimal performance can only be achieved by sacrificing the precision of the disseminated context. In the proposed approach, as will be seen, the context consumers play a decisive role in the process of context dissemination as well as the update rate of the relevant context data.

6.3 Cross-Domain Context Management

Basically, when a CS receives a query referring to an entity's context information stored in the local repository the procedure is straightforward. When the required context information is not stored in the local repository it has to be retrieved from a remote CS. An efficient look-up mechanism for finding this context information is essential for the scalability of the whole system. To achieve this mechanism, this thesis chooses to disseminate the context information to the HDS only when there is a consumer for this information. That is, this context information must have only one copy which must be published in the HDS. This choice is made for the following reasons:

(i) Efficient cross-domain query handling: having all context information related to an entity in one place (HDS) can be exploited during the query resolution phase in order for the applications to retrieve the context information more efficiently. That is, handling a query submitted to the system requires considering the context information in the entity's HDS disseminated from different domains instead of sending sub-queries to all VDSs. Thus, the querying response time decreases significantly.

(ii) Privacy ensuring: the alternative to publish the actual data at the HDS would be to only keep references to the relevant visited context server.

However, this weakens the privacy support as the context data is stored by the foreign domain that provides the sensor infrastructure. Thus there is a need to design a protocol between CSs which forces the context information to be centralized in the HDS. This way, enforcing user’s privacy policy will be feasible.

(iii) Cross-domain reasoning: it becomes possible to reason about the context information across different domains (e.g. tracking and understanding user’s tendency) and to identify the contextual situations which span different domains (see [119] for example). Moreover, this enforces the idea that each domain should have its own inference mechanism and in the home domain a cross-domain inference mechanism becomes possible.

(iv) High efficiency: it would be more efficient if we disseminate context to the HDS depending on how often the context change and at the same time on the context consumers needs. In the case of roaming users across domains, additional restrictions may arise (e.g. concerning the limited network connectivity, device power consumption, privacy enforcement, etc.), rendering imperative the need to establish an optimized mechanism in support of optimized context dissemination among domains taking into account the explicit requirements of consumers.

The following subsections present the designed and implemented middleware, *ubique*, which aims at optimizing and controlling the amount of exchanged context information in such a way that context information can efficiently and easily flow from context providers to consumers. *ubique* envisions a highly distributed and loosely coupled solution in order to exchange context information between context providers, CSs, and applications. Semantic meaning of the context information exchanged is added via distributed ontologies attached to it. Therefore, the *ubique* context management aims to: (i) enable the discovery of context providers, (ii) standardize context exchange between providers and consumers, (iii) disseminate contexts among CSs, (iv) share common understanding about context information elements, (v) standardize and enforce privacy, (vi) allow context providers to publish on demand where there is a consumer, (vii) relieve CSs from the burden introduced by frequent updates to the HDS, and (viii) prohibit overloading the context consumers with context information that does not interest them for the time being.

6.3.1 *ubique* Context Meta-Model

Context information can be represented in many ways. For *ubique* context modelling, the chosen approach is based on XML and makes use of ontologies that are described in OWL-DL for more detailed information about entities and their context types, as well as to support reasoning. As illustrated in Figure 6.1, the context information is represented in terms of context elements, which provide information about *context entities*, *context types* and *meta-data*.

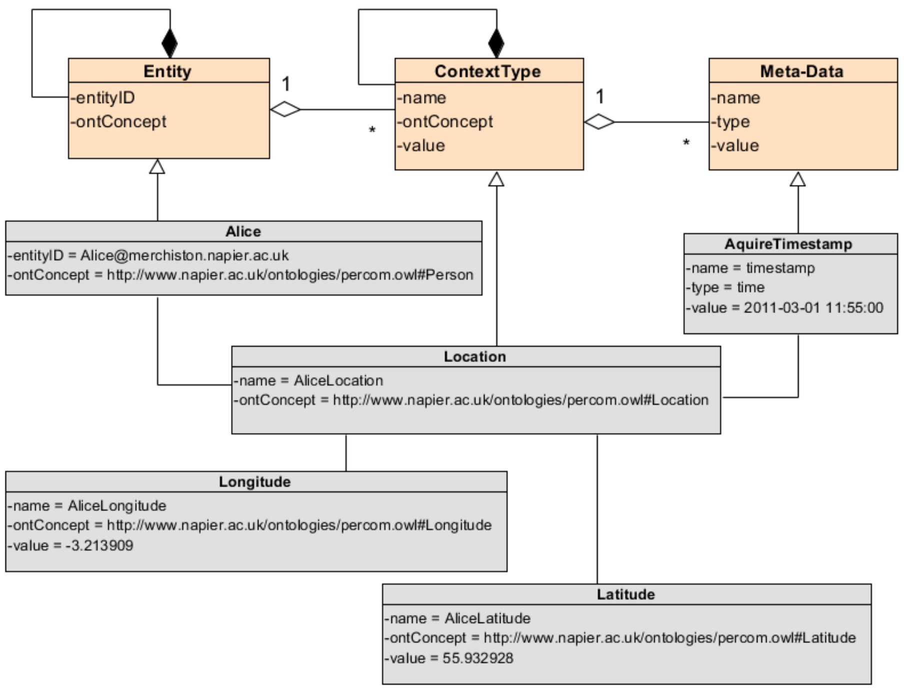

Entity
-entityID
-ontConcept
1
*
ContextType
-name
-ontConcept
-value
1
*
Meta-Data
-name
-type
-value
Alice
-entityID = Alice@merchiston.napier.ac.uk
-ontConcept = http://www.napier.ac.uk/ontologies/percom.owl#Person
AquireTimestamp
-name = timestamp
-type = time
-value = 2011-03-01 11:55:00
Location
-name = AliceLocation
-ontConcept = http://www.napier.ac.uk/ontologies/percom.owl#Location
Longitude
-name = AliceLongitude
-ontConcept = http://www.napier.ac.uk/ontologies/percom.owl#Longitude
-value = -3.213909
Latitude
-name = AliceLatitude
-ontConcept = http://www.napier.ac.uk/ontologies/percom.owl#Latitude
-value = 55.932928

Figure 6.1 The proposed context meta-model

The main assumption in the proposed model is the representation of relationships between entity and information: context entities (such as persons, places, events, etc.) are identified and classified by an ID and, optionally, a reference to an ontology concept representing them in order to establish a common understanding of the semantics of different entities in the pervasive environment. Each context entity is associated with a set of context types (such as address, location, etc.) which may include other context types. Further, each context type may be characterized by a set of metadata which contains, for example, source of information, timestamps, expiration time, and any Quality-of-Context information such as accuracy and confidence.

6.3.2 Context Management Components

The *ubique* context management middleware is designed for the discovery of, exchange of, and reasoning on context information across domains. It provides the relevant context information for the service or application, using distributed sensing infrastructure and centralized storing mechanisms. *ubique* is defined as a set of components which are loosely coupled to provide relevant context information both by sensing and interpreting mechanisms. These key components or building blocks are depicted in Figure 6.2, and described below.

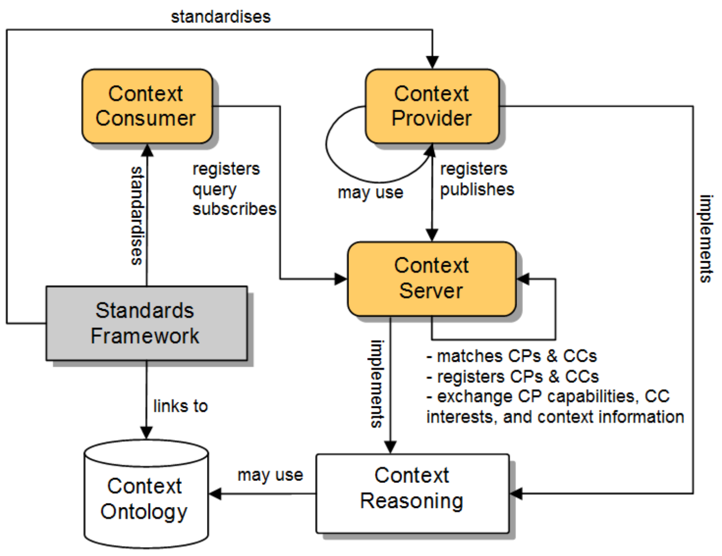


Figure 6.2 *ubique* components

Context Consumer: (CC) is a software entity that uses the CS interface to register its context interest or query. The CC receives the requested context information asynchronously by submitting context interest and synchronously by submitting context query to the CS. A CC exposes interfaces to start receiving context information from the corresponding CS when they become available. These interfaces adhere to standards defined in the Standards Framework (SF).

Context Provider (CP): is a software entity that uses the CS interface to register its capability of providing context information. A CP exposes interfaces to publish context information to the corresponding CS on-demand. These interfaces adhere to standards defined in the SF. It is registered in the CS so that context consumers can discover and introspect it. Note that any software agent, reasoner, or storage component can be a CP as long as it adheres to the interfaces defined in SF. Usually, CPs wrap context sources such as GPS receiver or temperature sensor to provide their information.

Context Server (CS): provides a registration service for CPs to register/update/unregister their capabilities that uniquely describe their functionalities and for CCs to register/update/unregister their context interests that can be matched against the available CPs, and enables the discovery of various context providers. Additionally, it provides services to exchange the CCs' context interests and CPs' capabilities between CSs as will be seen later.

Standards Framework (SF): A set of specifications describing the CP capabilities, the CC interests and queries, the interfaces to exchange commands and context information between different components, a format to exchange an atomic context information element, as well as a format for privacy tags.

ubique relies on the reasonable assumption that a CS is identified by its Internet domain name and that the CS is responsible for managing the context information available in its domain. Additionally, each entity (sensor, user, application, etc.) has a unique ID that should be registered in one of the CSs. For example Alice ID could be `Alice@merchiston.napier.ac.uk` as she is a registered user in the CS of the domain `merchiston.napier.ac.uk` which is Alice's HDS.

The Context Ontology (CO) describes the logical relations between the different context concepts in OWL-DL. This ontology is used to get more detailed information about context types and entities, as well as to support the Context Reasoning process.

6.3.3 Context Interfaces and Operations

ubique provides three different interfaces which allows the integration of CSs, CCs, and CPs into the eco-system. In the following the main interfaces and the main corresponding operations are described.

a. Integrating Context Providers: The provided operations allow registering CPs and their information with the CS as well as providing a discovery function through which participating components can check for available CPs.

registerContextProvider: This operation is used by the CP to advertise its capabilities in terms of the types of context information it can provide and the relevant entities playing a role in this information. Additionally, the registration provides a set of available CP meta-data (describing the CP and the quality of context information it provides). For example, the user's location can be measured with different qualities by location sensors like GPS, CellId, WLAN-in-range, etc. The CP capabilities XML scheme is depicted in Figure 6.3.

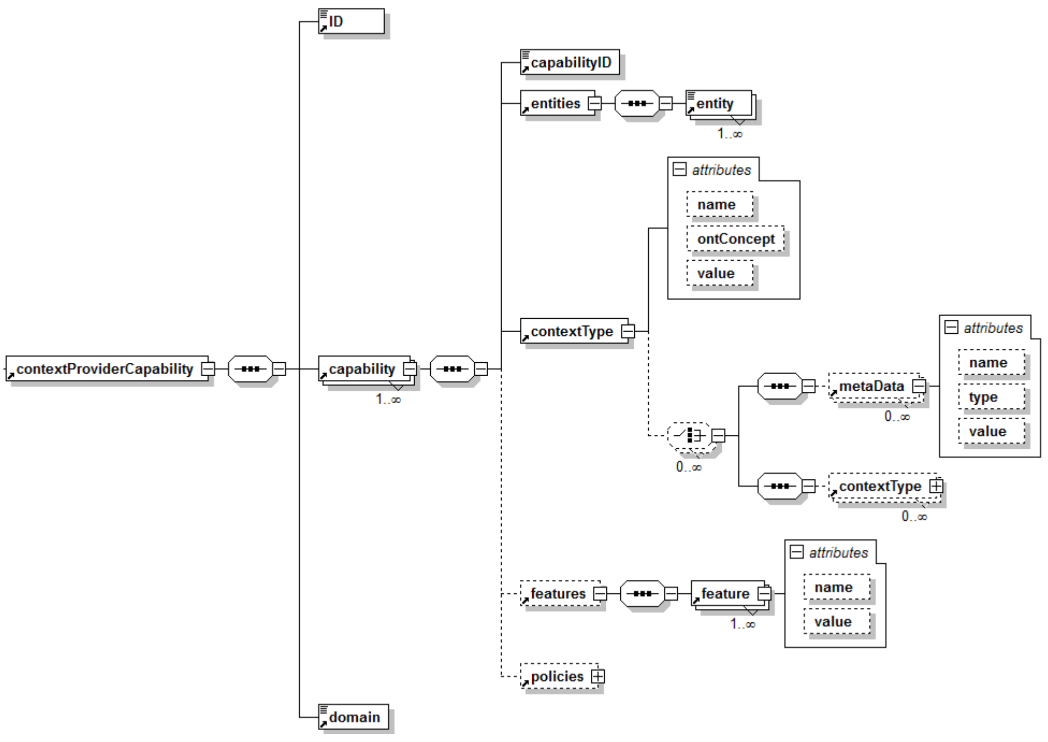


Figure 6.3 CP capabilities XML scheme

Basically, the CP specifies in its capabilities its ID, the domain its information is originated from, and one or more *capabilities*. Each *capability* specifies its

ID, the entities playing a role in the context information the CP can provide, and the supported context types. Optionally, it specifies the meta-data about these context types, its different attributes (features), and collection policies.

discoverContextProviders operation is used by the CCs to get the list of available CPs and their capabilities for later query.

sendCPCommand: This operation is used by the CS to command a specific CP to start/stop publishing its information. The command message contains a reference (tuple ID) where the context information should be pushed.

b. Integrating Context Consumers: The provided operations allow registering CCs with the CS, querying (synchronously), as well as subscribing in order to be notified about context information (asynchronously).

queryContextServer: This operation is used by the CC to synchronously request context information. The CC specifies its interest in terms of the needed context types of specific entity(ies), as well as additional constraints on the CPs and context types meta-attributes.

subscribeContextConsumer: This operation enables long-lasting monitoring of the system. Basically, the logic of this operation is similar to the latter operation, but the requested context information is returned in the form of an asynchronous “notify” callback operation. Figure 6.4 depicts the CC interest XML scheme. The CC can specify one or more *interests*. Each *interest* specifies its ID, the entities the CC is interested in to get their context information, and the interested context types. Optionally, it specifies the condition(s) on the context types, the domain(s) this information is originated from, the CP’s required feature(s), and the ID of a specific CP.

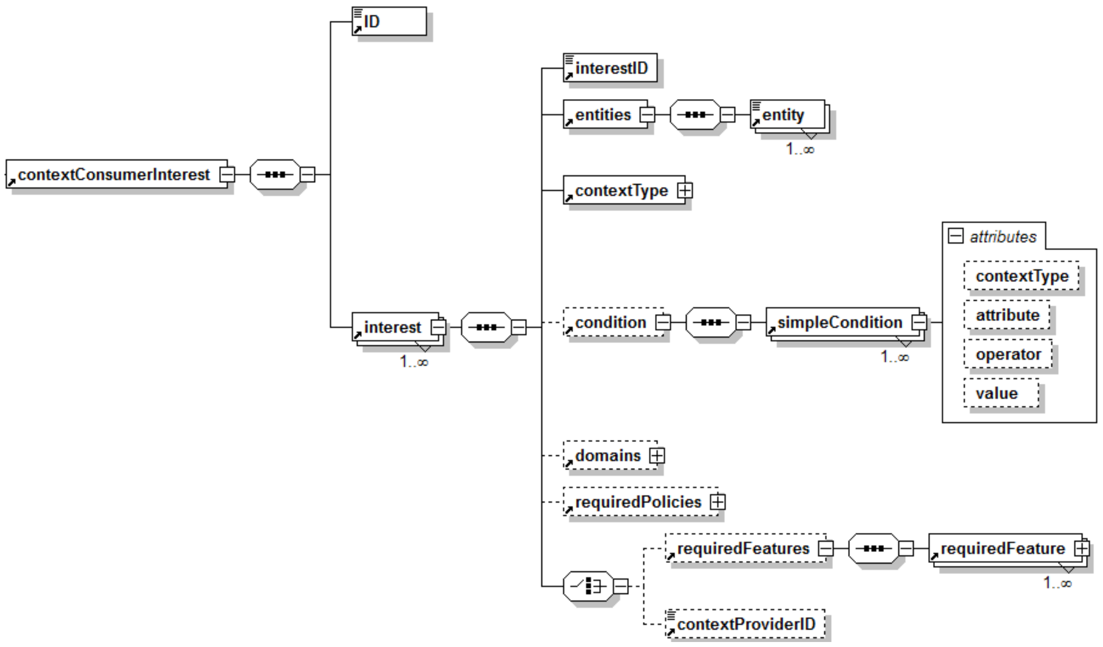


Figure 6.4 CC interest XML scheme

sendCCCommand: This operation is used by the CS to command a specific CC to start/stop receiving the information it has subscribed to. The command message contains a reference (tuple ID) where the context information should be popped.

c. Collaboration between CSs: as already mentioned, every CS is responsible of providing and storing context information related to entities registered in it. Since the sensor infrastructure in each domain may provide context information about roaming entities, a collaboration protocol is needed between CSs in order to disseminate this information to the entities' HDSs. Three types of information exchanged between CSs can be distinguished:

- CP Capabilities: CPs may advertise their ability to provide context information about entities not registered in the current domain. For example, when Alice moves from her home domain (`domain1.com`) to `domain2.com`, a location provider (a registered entity in `domain2.com`) can advertise its ability to provide the location information about `Alice@domain1.com` to the CS of `domain2.com`. In this case, the CS of `domain2.com` should disseminate the CP capability to `domain1.com` (Alice's HDS) which is responsible to handle all queries related to Alice.

- CC Interests: A CS may receive context interest about entities not registered in it. In this case, the CS should disseminate these interests to the HDS of the corresponding entities.

- Context information: The idea is that each CS has to maintain a repository for all CP capabilities able to provide context information about its registered entities as well as all CC interests related to these entities. Any change in this repository (i.e. addition, updating, or deletion of a CP capability or CC interest) should trigger a matching function which tries to bind a CP with a CC. When a match is found, (i) a new tuple has to be created; (ii) a *startPublishing* command message has to be sent to the CP (via *sendCPCommand* operation) along with the corresponding CC interest and tuple ID; and (iii) a *startReceiving* command has to be sent to the CC (via *sendCCCommand* operation) along with the tuple ID. The CP now has all the information necessary to know what kind of context types, for which entities, and when to publish to the tuple (e.g. regularly or for a context changes greater than a specific threshold, etc.). Note here that when, for example, an application is interested in Alice location in `domain2.com`, the CS of `domain1.com` (Alice's HDS) will create a tuple in CS of `domain1.com` and command the CP of Alice location to start publishing in this tuple. In other words, all the context information related to Alice, even those emerging from foreign domains, will be kept in her HDS. This way, the user's privacy can be enforced. This mechanism is illustrated in the case study in Section 8.4.

Figure 6.5 depicts the XML scheme of the published context information which we call a *contextlet*. Basically, each *contextlet* specifies the CP ID, the interest ID, the domain from which this information is originated, the entity in question, and the list of the requested context types and their values.

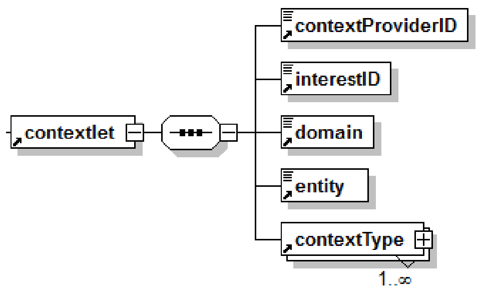


Figure 6.5 Contextlet XML scheme

6.3.4 Privacy

Privacy is about protecting users' personal information, which may include also context information e.g. location, mood, etc. In the *ubique* approach, to ensure the confidentiality of the privacy-sensitive information, users have the flexibility to define their own privacy policy covering all types of context information that may be distributed in different domains.

Obviously, the sensor infrastructure in each domain may report context information related to entities out of the scope of the current domain which in turn weakens the privacy ensuring mechanism and loosens control over the context originated in different domains. In this case as aforementioned the context information of the foreign entities must be published in their HDS with the following conditions: (i) there is a corresponding consumer for this information, and (ii) revealing this information does not violate the privacy policy of the corresponding entity. If the request (query) does not violate the privacy policy then the CS commands the CP to start publishing the required context information at the entity's HDS; otherwise, an "access denied" response is sent to the CC. Figure 6.6 shows the privacy tag scheme used in *ubique*. Each user (or each entity in general) has the flexibility to specify its privacy policy covering the context types and the domains containing the context information. The *privacyTag* specifies for each context type the CCs having the right to get access to the context information and the time intervals during which this context information can be revealed to them.

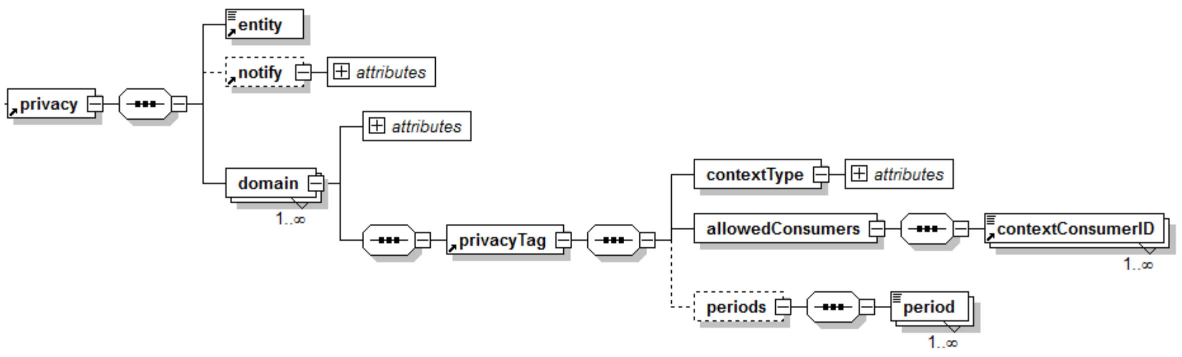
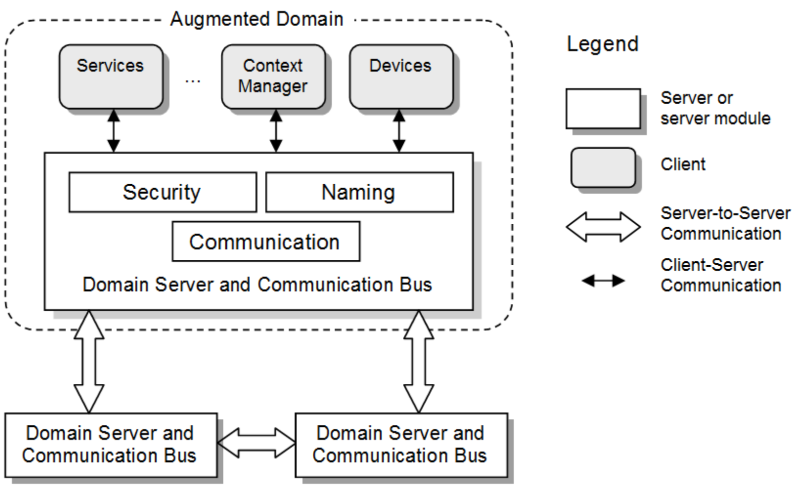


Figure 6.6 Privacy XML scheme

Finally, secure storage of context information requires proper authentication and authorization to access it. Therefore, each CC is assumed to be a computational entity registered in one of the CSs which means that it has a unique ID and password, and it must be authenticated by its CS.

6.4 *ubique* Implementation

Figure 6.7 illustrates the proposed domain-based context-aware computing eco-system. In general, the system should integrate distributed hardware and software components and provide a naming scheme for those entities. The eco-system starts from a single system with client-server architecture; then multiple systems federate together through server-to-server communication to form the eco-system. A single system usually manages local clients, such as users and devices in a specific domain.



Augmented Domain
Services
...
Context Manager
Devices
Security
Naming
Communication
Domain Server and Communication Bus
Domain Server and Communication Bus
Domain Server and Communication Bus
Legend
Server or server module
Client
Server-to-Server Communication
Client-Server Communication

Figure 6.7 Domain-based context-aware eco-system

The server is called Domain Server and Communication Bus. The server provides core functionalities, such as security and naming, and acts as a communication infrastructure for clients available in its administrative domain. The naming scheme is similar to that of e-mail systems. Each server has a unique domain name; clients have their names concatenated to the server name. Clients from different systems can also communicate with each other with the server-to-server communication. Clients could be devices, such as sensors, and applications that provide services to the user. Clients can be also services that provide functionalities the server does not provide such as the context manager (see Figure 6.7). Clients have to be authenticated by the server to use the system.

Notice that the server does not provide the context management service itself, leaving that responsibility to a separate client, the context manager. The context manager can be easily replaced or upgraded without affecting the whole system. The client-server and server-to-server communication interfaces are standardized, which facilitates the system extensibility.

In order to robustly implement the *ubique* approach, relying on a standard or already established protocol is obviously a preferred choice. As aforementioned in Section 2.3.4, the eXtensible Messaging and Presence

Protocol (XMPP) [145] (also known as a Jabber protocol) is widely adopted open protocol for instant messaging and is designed for near real-time communication.

6.4.1 Jabber and Domain-based Context Management

As aforementioned, the proposed domain-based context management middleware is based on Jabber technologies. Jabber has been chosen because its design, architecture, and features match our requirements: In the pervasive environment the interaction between different entities should be generic and not in a particular format. Jabber provides a rich set of communication mechanisms. Moreover, the context management infrastructure should support the interaction between different users, devices, and software components in a universal way. In Jabber systems, any entity that implements the XMPP-Core and its extensions protocols can establish a connection with a Jabber server and interact with other entities on any Jabber server. Thus the open architecture and standardization of the Jabber platform ease its adoption to build *ubique*.

Apart from these capabilities, Jabber has other advantages such as its increasing popularity and community support; the availability of a set of servers, clients, and software libraries supporting a low-barrier entry for developers; and its adoption of XML to communicate messages between entities make it possible to use existing XML tools and libraries.

6.4.2 Jabber and Context Manager

Jabber entities can be implemented either as clients or as external server components. Clients use the protocols defined in “XMPP Core” to connect to the Jabber server; external components use the “Jabber Component Protocol” (JCP) [XEP-0114] for the connection. These two types of entities are functionally similar; thus for a given service, we can implement it as either a client or a component. However, unlike client components whose contact lists and subscription are maintained by the Jabber server, an external component has to manage its subscriptions and contact lists by itself. The naming convention for external components is different from client components. For example, the context manager JID might be `context@merchiston.napier.ac.uk` if it is implemented as a client, and `context.merchiston.napier.ac.uk`, if it is implemented as an external component.

In *ubique* the context manager has been implemented as an external Jabber component. The choice of considering the context manager as an extension to the Jabber server functions is more of design decision than a functional one. Figure 6.8 shows the architecture of the context manager: *ContextMgr*. The PubSub server is also a Jabber component. ContextMgr component connects to a Jabber server using JCP. The ontology that describes the context is stored in a Web server. The actual context data (contextlets) is stored in the PubSub so that the PubSub server can notify the subscriber of any context changes.

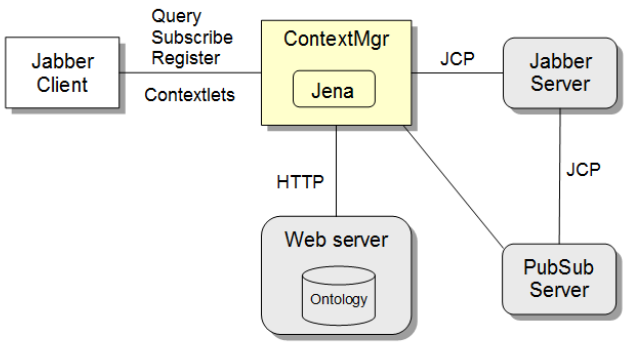


Figure 6.8 The context manager external component

In Figure 6.9, two Jabber servers are inter-connected; one of them connects to a CP and the other connects to a CC. The context manager, *ContextMgr*, connects to the Jabber server as a Jabber external component. The continuous lines represent the transport connections which are the actual routes for transferring data. On the other hand, the dashed lines indicate logical connections which means the communication between two end points does not happen directly, but through physical ones.

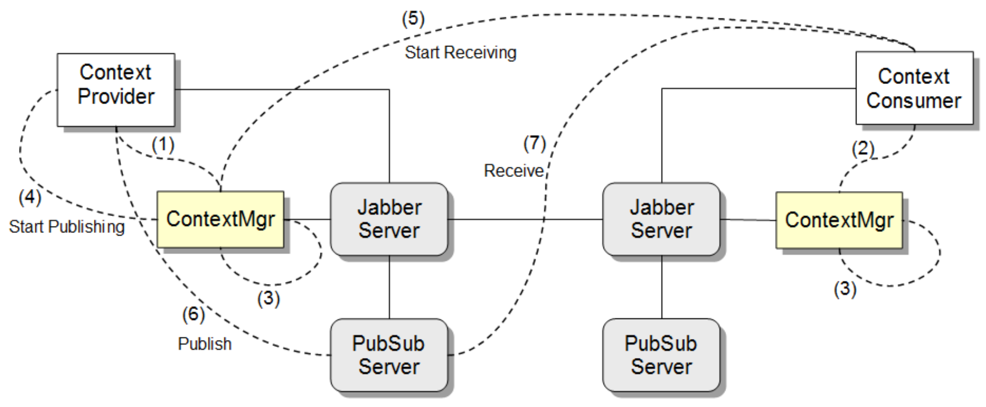


Figure 6.9 *ubique* components interactions

When the system starts up, both CP and CC login to their Jabber servers. Then, the capabilities of each CP and the interests of each CC are registered with the corresponding Jabber server (Step 1 and 2). Thus the context manager can match the published CPs' capabilities with the CCs' interests or queries (Step 3). If the context manager decides that the CC interest matches the CP capability and this does not violate any entity's privacy, then it creates a tuple space in the local PubSub server and sends the *startPublishing* command message to the CP (Step 4) and the *startReceiving* command message to the CC (Step 5) along with the tuple space ID embedded in the message. Once the CP publishes a new contextlet (Step 6), the CC can receive it asynchronously (Step 7). For the CC query, when the context manager decides which CP can have the requested context information it queries that CP and returns the result to the CC synchronously.

In *ubique*, the OpenFire [146] has been used as a XMPP server, and the context manager has been implemented in Java. *ubique* aims to achieve the goal of controlling the context information dissemination between administrative domains in a way that is efficient in terms of saving network bandwidth and devices energy, as well as respecting people privacy in the pervasive environment. The system has a clear architecture and is highly extensible.

6.5 Case Study: Smart University System

6.5.1 Objective

This case study illustrates, verifies and evaluates the use of *ubique* middleware for context dissemination between different context servers distributed in different domains. It shows also how the developers can specify CCs' queries and CPs' capabilities, and how users can specify their privacy policies.

The *ubique* approach has been realized in one scalable real-life application. Edinburgh Napier University had the ambition to build an ICT-driven Smart University system; part of the scheme is to provide cross-campus real-time virtual collaboration between working groups of staff and students, such as team members working on a research project, students doing a group project and committee members within a school, faculty or even the whole university. University staff and students roam among campuses, and experience different activities. This *ubique*-enabled system can be used by members of the above groups to keep updated about each other's current activities, status and interests, and to exchange information so that they can avoid disturbing and interact more intelligently.

Here one scenario from the Smart University system has been taken to demonstrate how *ubique* approach and the system work. Alice and Bob are professors working on an EPSRC-sponsored research project. They are both based at the Merchiston campus of Edinburgh Napier University. Alice has a post-doc, Carol, who is a research assistant on the project and needs to travel among the campuses for her research. Alice would like to keep updated about Bob's activities and Carol's location.

6.5.2 Solution and Implementation

Different components could be identified in this scenario: The context server available in Merchiston campus (`merchiston.napier.ac.uk`), the context server available in Sighthill campus (`sighthill.napier.ac.uk`), the context provider which provides information about the activities of entities located in Merchiston campus, the context provider which provides the location information of entities available in Sighthill campus, and the application itself which is considered here as an entity registered in the context server `merchiston.napier.ac.uk`.

Figure 6.10 depicts the sequence of exchanging information between different components: CPs, CCs, and CSs.

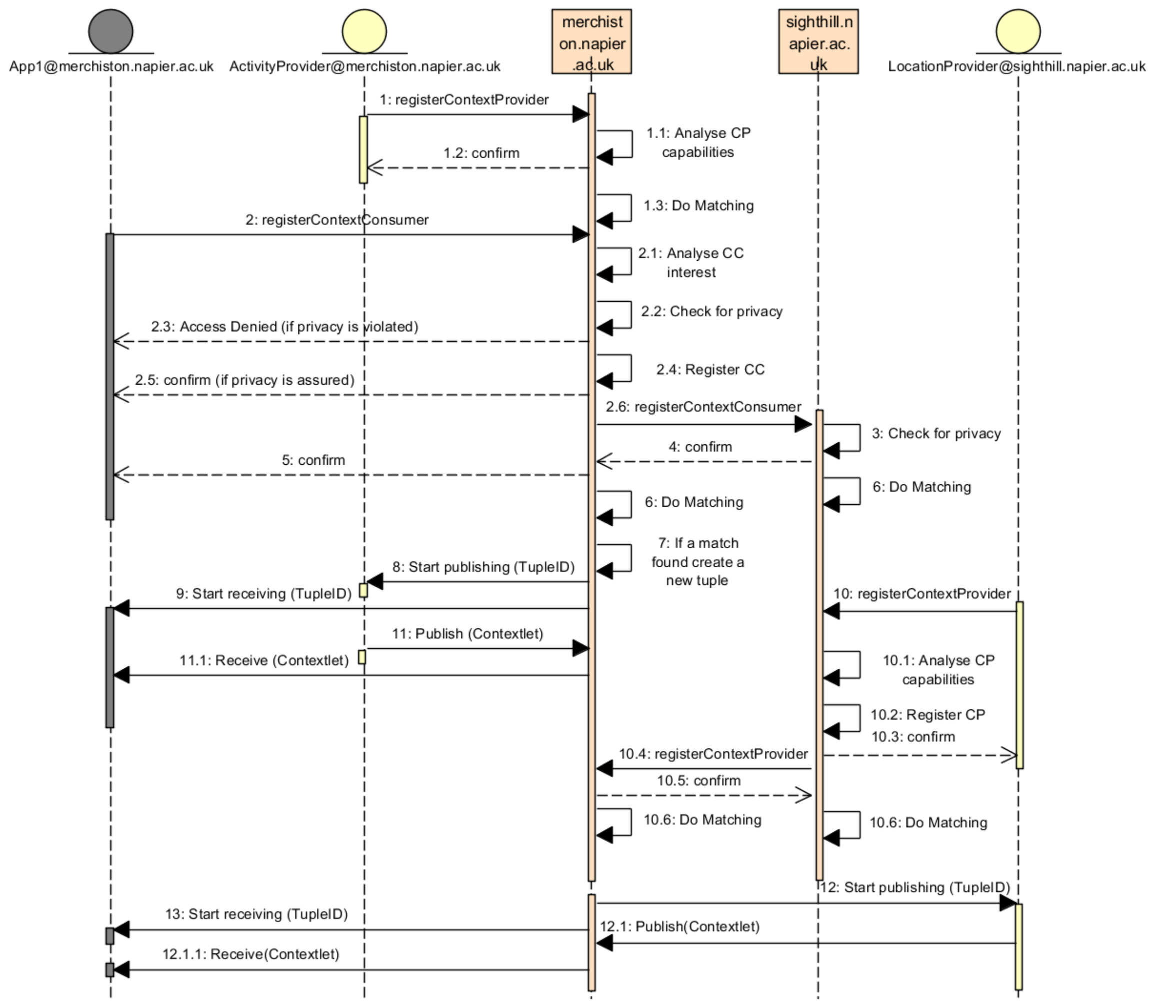
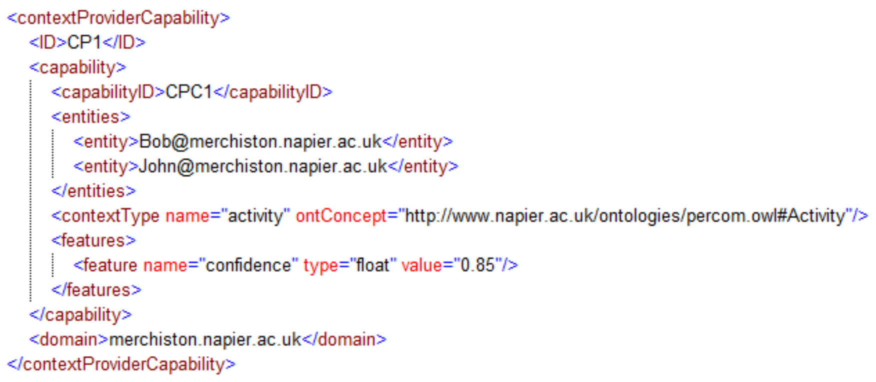

App1@merchiston.napier.ac.uk
ActivityProvider@merchiston.napier.ac.uk
merchiston.napier.ac.uk
sighthill.napier.ac.uk
LocationProvider@sighthill.napier.ac.uk
1: registerContextProvider
1.1: Analyse CP capabilities
1.2: confirm
1.3: Do Matching
2: registerContextConsumer
2.1: Analyse CC interest
2.2: Check for privacy
2.3: Access Denied (if privacy is violated)
2.4: Register CC
2.5: confirm (if privacy is assured)
2.6: registerContextConsumer
3: Check for privacy
4: confirm
5: confirm
6: Do Matching
6: Do Matching
7: If a match found create a new tuple
8: Start publishing (TupleID)
9: Start receiving (TupleID)
10: registerContextProvider
11: Publish (Contextlet)
11.1: Receive (Contextlet)
10.1: Analyse CP capabilities
10.2: Register CP
10.3: confirm
10.4: registerContextProvider
10.5: confirm
10.6: Do Matching
10.6: Do Matching
12: Start publishing (TupleID)
13: Start receiving (TupleID)
12.1: Publish(Contextlet)
12.1.1: Receive(Contextlet)

Figure 6.10 Interaction between different components

This is described as follows: The CP `ActivityProvider@merchiston.napier.ac.uk` registers the following capability in its HDS and wait for confirmation (Step 1).

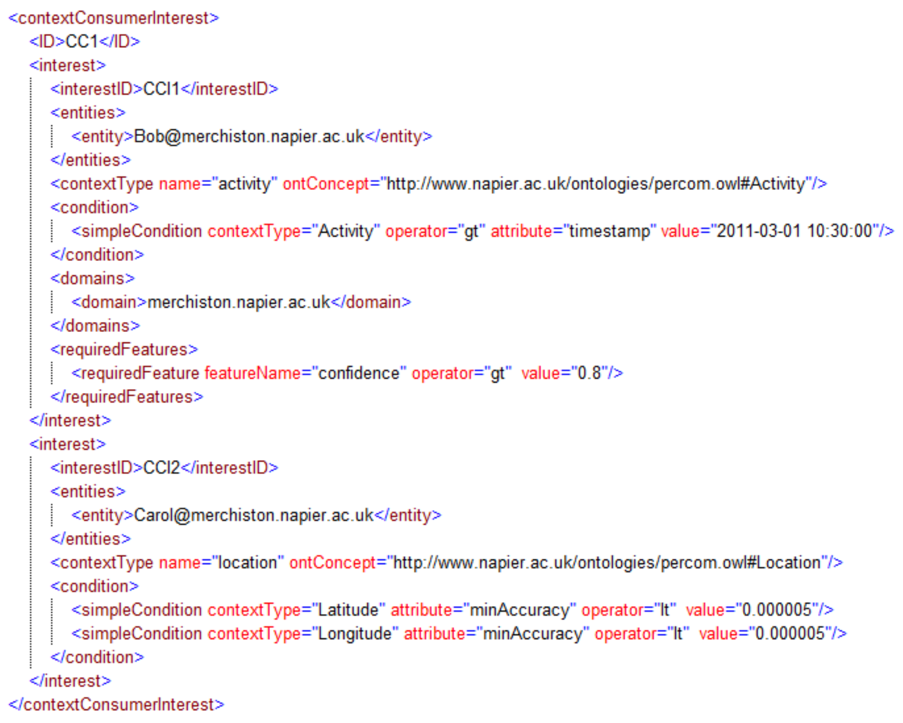


```
<contextProviderCapability>
  <ID>CP1</ID>
  <capability>
    <capabilityID>CPC1</capabilityID>
    <entities>
      <entity>Bob@merchiston.napier.ac.uk</entity>
      <entity>John@merchiston.napier.ac.uk</entity>
    </entities>
    <contextType name="activity" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Activity"/>
    <features>
      <feature name="confidence" type="float" value="0.85"/>
    </features>
  </capability>
  <domain>merchiston.napier.ac.uk</domain>
</contextProviderCapability>
```

Figure 6.11 Example of the activity provider's advertised capability

The CS analyzes the received CP capability to see if any of the supported entities is not registered in it. Because this CP does not provide context information about entities not registered in `merchiston.napier.ac.uk` no further interaction with other CSs has to be taken. Obviously, any change in the available CPs or CCs triggers the matching function.

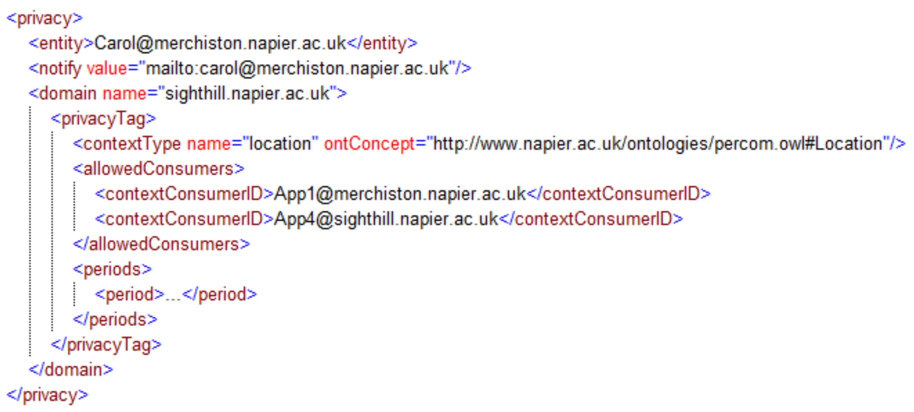
For the sake of simplicity and without loss of generality, the example application `App1@merchiston.napier.ac.uk` is registered in Alice's HDS. It registers the following CC interest (Step 2):



```xml
<contextConsumerInterest>
  <ID>CC1</ID>
  <interest>
    <interestID>CCI1</interestID>
    <entities>
      <entity>Bob@merchiston.napier.ac.uk</entity>
    </entities>
    <contextType name="activity" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Activity"/>
    <condition>
      <simpleCondition contextType="Activity" operator="gt" attribute="timestamp" value="2011-03-01 10:30:00"/>
    </condition>
    <domains>
      <domain>merchiston.napier.ac.uk</domain>
    </domains>
    <requiredFeatures>
      <requiredFeature featureName="confidence" operator="gt"  value="0.8"/>
    </requiredFeatures>
  </interest>
  <interest>
    <interestID>CCI2</interestID>
    <entities>
      <entity>Carol@merchiston.napier.ac.uk</entity>
    </entities>
    <contextType name="location" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Location"/>
    <condition>
      <simpleCondition contextType="Latitude" attribute="minAccuracy" operator="lt"  value="0.000005"/>
      <simpleCondition contextType="Longitude" attribute="minAccuracy" operator="lt"  value="0.000005"/>
    </condition>
  </interest>
</contextConsumerInterest>
```

Figure 6.12 Example of an application's context interest

This CC interest shows that the application is interested to know the location of Carol in any domain and the activity of Bob in the `merchiston.napier.ac.uk` domain. Note here that any CP registered in `merchiston.napier.ac.uk` domain or in any of its sub-domains is eligible to be matched with the interest CCI1. For each context interest, the CS checks for the corresponding entity privacy before registering it. Figure 6.13 shows an example of Carol privacy tag.

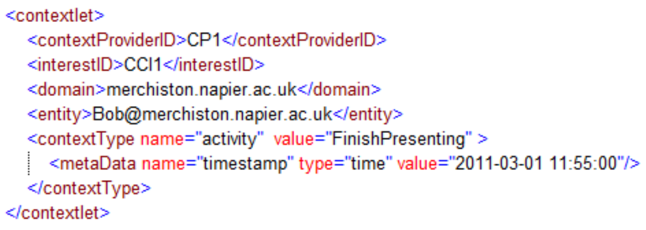


```xml
<privacy>
  <entity>Carol@merchiston.napier.ac.uk</entity>
  <notify value="mailto:carol@merchiston.napier.ac.uk"/>
  <domain name="sighthill.napier.ac.uk">
    <privacyTag>
      <contextType name="location" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Location"/>
      <allowedConsumers>
        <contextConsumerID>App1@merchiston.napier.ac.uk</contextConsumerID>
        <contextConsumerID>App4@sighthill.napier.ac.uk</contextConsumerID>
      </allowedConsumers>
      <periods>
        <period>...</period>
      </periods>
    </privacyTag>
  </domain>
</privacy>
```

Figure 6.13 Example of a privacy policy

If the privacy is violated, an “access denied” message should be sent to the application; otherwise the context interest will be registered and a confirmation message should be sent to the application.

The CS of `merchiston.napier.ac.uk` finds out that there is a match between the CP capability whose ID is CPC1 (Figure 6.11) and the CC interest whose ID is CCI1 (Figure 6.12), therefore, it creates a tuple and sends the necessary commands so that `ActivityProvider@merchiston.napier.ac.uk` starts publishing contextlets in the created tuple and `App1@merchiston.napier.ac.uk` starts receiving the published contextlets. Figure 6.14 shows an example of the contextlet sent by the activity provider. Alice may like to send Bob a congratulations message when he finishes his presentation.



```
<contextlet>
   <contextProviderID>CP1</contextProviderID>
   <interestID>CCI1</interestID>
   <domain>merchiston.napier.ac.uk</domain>
   <entity>Bob@merchiston.napier.ac.uk</entity>
   <contextType name="activity"  value="FinishPresenting" >
      <metaData name="timestamp" type="time" value="2011-03-01 11:55:00"/>
   </contextType>
</contextlet>
```

Figure 6.14 Example of contextlet received from activity provider

In `merchiston.napier.ac.uk` there is no provider for Carol location. When Carol roams to `sighthill.napier.ac.uk` the CP `LocationProvider@sighthill.napier.ac.uk` reports its ability (Figure 6.15) to provide Carol as well as other entities locations to CS of `sighthill.napier.ac.uk`.

```
<contextProviderCapability>
  <ID>CP2</ID>
  <capability>
    <capabilityID>CPC1</capabilityID>
    <entities>
      <entity>Carol@merchiston.napier.ac.uk</entity>
      <entity>Sally@sighthill.napier.ac.uk</entity>
    </entities>
    <contextType name="location" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Location">
      <contextType name="latitude" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Latitude">
        <metaData name="minAccuracy" type="float" value="0.000002"/>
      </contextType>
      <contextType name="longitude" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Longitude">
        <metaData name="minAccuracy" type="float" value="0.000002"/>
      </contextType>
    </contextType>
  </capability>
  <domain>sighthill.napier.ac.uk</domain>
</contextProviderCapability>
```

Figure 6.15 Example of the location provider advertised capabilities

The CS of `sighthill.napier.ac.uk` finds out that the location provider is able to provide Carol location which is not registered in it; thus, it disseminates the CP capability depicted in Figure 6.16 to Carol HDS: `merchiston.napier.ac.uk` (Step 10.4 in Figure 6.10). Notice that this capability is the same of Figure 6.15 except that the entities not registered in `merchiston.napier.ac.uk` have been removed.

```
<contextProviderCapability>
  <ID>CP2</ID>
  <capability>
    <capabilityID>CPC1</capabilityID>
    <entities>
      <entity>Carol@merchiston.napier.ac.uk</entity>
    </entities>
    <contextType name="location" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Location">
      <contextType name="latitude" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Latitude">
        <metaData name="accuracy" type="float" value="0.000002"/>
      </contextType>
      <contextType name="longitude" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Longitude">
        <metaData name="accuracy" type="float" value="0.000002"/>
      </contextType>
    </contextType>
  </capability>
  <domain>sighthill.napier.ac.uk</domain>
</contextProviderCapability>
```

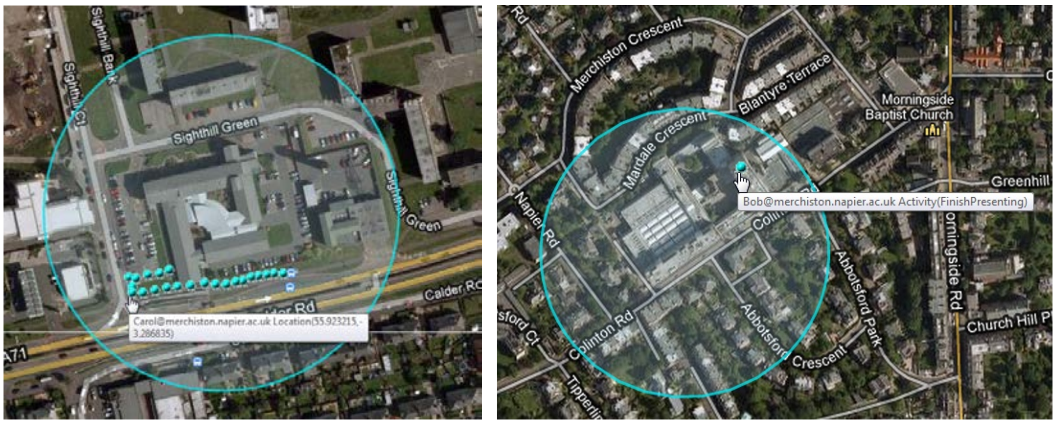
Figure 6.16 The location provider capabilities disseminated to Carol HCS

After the re-matching process, the CS of `merchiston.napier.ac.uk` finds out that there is a CP able to provide Carol's position. Therefore, as in the previous case, it creates a tuple and sends the necessary commands to the corresponding entities; however, this time the locally published contextlets are pushed by a CP from other domain. Figure 6.17 shows an example of a contextlet published by the location provider indicating Carol's location.

```xml
<contextlet>
   <contextProviderID>CP2</contextProviderID>
   <interestID>CCI2</interestID>
   <domain>sighthill.napier.ac.uk</domain>
   <entity>Carol@merchiston.napier.ac.uk</entity>
   <contextType name="location" >
      <metaData name="timestamp" type="time" value="2011-03-01 11:55:00"/>
      <contextType name="latitude" value="55.923215" >
         <metaData name="timestamp" type="time" value="2011-03-01 14:15:00"/>
         <metaData name="acurracy" type="float" value="0.0000002"/>
      </contextType>
      <contextType name="longitude" value="-3.286835">
         <metaData name="timestamp" type="time" value="2011-03-01 14:15:00"/>
         <metaData name="acurracy" type="float" value="0.0000002"/>
      </contextType>
   </contextType>
</contextlet>
```

Figure 6.17 Example of Carol location contextlet

Figure 6.18 depicts screenshots of the example application. The cyan circles represent roughly the domain border of each CS. Each small dot circle represents a contextlet.


Carol@merchiston.napier.ac.uk Location(55.923215,-3.286835)

Bob@merchiston.napier.ac.uk Activity(FinishPresenting)

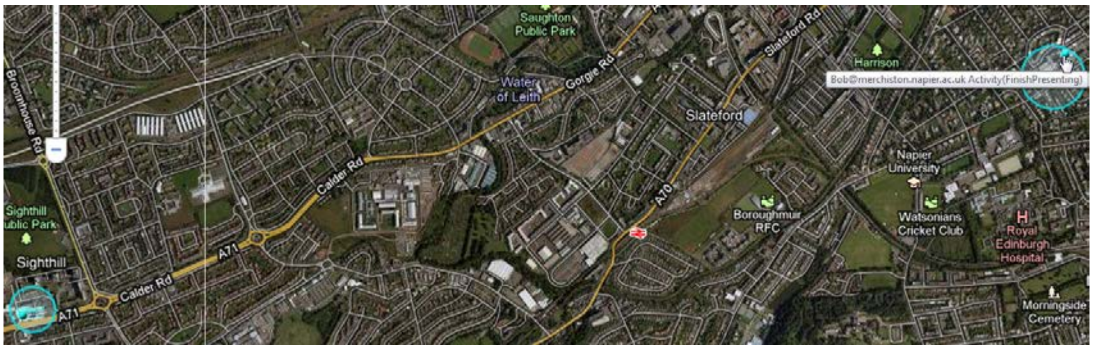


Figure 6.18 Screenshots of the example application

6.5.3 Summary

This case study has illustrated the usage of the *ubique* middleware to hide the increasing complexity of context management available in different domains from applications. Developers are able to retrieve the interested context information originated from different spatial domains by specifying domain-based context queries and interests. Thus, they are alleviated from finding out which repository (CS) has the context information they need as well as what context providers capable to deliver this information.

The use of the standard API and schema for the contextlet, CC interests and queries, and CP capabilities makes it possible for any component to easily integrate the eco-system. The component has to be a registered entity in one of the available Jabber servers (CSs) which sometimes prohibit the spontaneous interaction between the CS and new entities. However, this requirement is in alignment with the need to enforce the user privacy and to disclose their context information only to already-known entities.

In a previous work [147], an evaluation of the *infinitum* middleware (former version of *ubique*) has been conducted in terms of the time required for disseminating contextlets between two CSs. The simulation has shown that disseminating 100 contextlets simultaneously requires a latency of around 1.8s which is acceptable for a wide-range of applications requiring dynamic context information e.g. position. However, the results are probably not quite representative as the latency is dominated by the actual cross-domain network bandwidth as well as the contextlets compression method used if any.

One of the main advantages of *ubique* is the enforcement of the user's privacy policy spanning different domains. In this respect, for example, Carol is able to specify the entities eligible to access her context information and during which periods. This allowed Carol for example to create and define one privacy policy and publish it in her HDS and thus alleviating her from defining several policies for different domains.

6.6 Conclusion

The essence of context-awareness is to let applications and users take full advantage of the available context information e.g., users' or devices' locations. The requirement for universal context access demands for a middleware solution as an essential requirement for building context-aware systems. Therefore, it is essential to establish innovative data storage and dissemination mechanisms. The architecture of *ubique* presented in this chapter hides the increasing complexity of context management from applications and incorporates advanced mechanisms that support mobile users.

The contribution of this chapter lies in the design and implementation of a distributed context management middleware and the associated context information dissemination protocol that addresses the requirements of scalable distributed context management, privacy enforcement, and efficient context information dissemination and query handling. In *ubique*, the storage and dissemination of the context information is performed between distributed CSs. *ubique* brings several unique features to cross domain context management as discussed in Section 9.1.2, all of which have been verified by case studies. The following chapter aims at taking advantage of the *ubique* infrastructure to capture and reason about the contextual situations that span one or more domains.

Chapter 7 Contextual Situation Recognition with Process Mining Techniques

This chapter first provides a formalization of the situation recognition problem and then focuses on the potential use of process mining techniques for measuring situation alignment, i.e., comparing the real situations of users with the expected situation models. To this end, two methods have been proposed to create and/or maintain the fit between them: LTL analysis and conformance testing. The effectiveness of the approach has been evaluated in Section 8.3 using a third party published smart home dataset. The experiments prove the effectiveness of applying the proposed approach to recognizing situations in the flow of context information.

7.1 Introduction

Situation awareness is the capability of the entities in pervasive computing environments to be aware of situation changes and automatically adapt themselves to such changes to satisfy user requirements, including security and privacy [12].

Following Dey's context definition, situation is a central notion describing context. Dey [23] defines situation as a "description of the states of relevant entities". As aforementioned in section 2.1.5, situations inject meaning into the application and are more stable, and easier to define and maintain than basic contextual facts. Thus, adaptations in context-aware applications are usually caused by the change of situations. A situation represents the semantic interpretation of context, and is generally derived by combining several pieces of low-level contexts in some way [148], with potentially many different contexts being indicative of the same situation. The situation notion permits a higher-level specification of human behaviour in the scene [9].

A natural way to program context-aware behaviour is to use rules which map a recognized situation to some given action. But a preceding question is how does one describe and represent the situations that a context-aware system should recognize? Is it enough to describe the current states, or it is necessary to consider the previous states? For instance, if we were building a context-aware application to recognize the different situations in a conference room, we would like the application to behave appropriately in certain situations – the application could somehow detect a situation via some combination of sensors and then adjust the camera direction accordingly. One could enumerate a set of typical situations (or situation types) which we are interested in (e.g. the speaker is talking, an audience is asking, one person is entering, etc.) and have rules to act appropriately in those situations. In this case it is enough to have some rules to represent these typical situations in terms of states inferred from sensor readings. However, for complex situations that call for tracking a user's behaviour we may need to consider the user's recent state history. For example, deciding on the appropriate service delivered to a user sitting in the living room depends on whether he is studying or he has just arrived from work.

Moreover, because context information is naturally distributed in different domains (areas), understanding the user's current situation may require considering the different states the user experienced in these domains. For example, to identify if the current day was busy for the user we need to consider the different activities and states the user has experienced in work, shopping, on the road, etc. Unlike some existing context-aware systems which isolate one context state from another or do not consider context states identified in different domains, this thesis aims at taking advantage of *ubique* to capture and reason about the different contextual situations spanning one or more domains.

On the other hand, in pervasive environments, both mobility and ubiquity are supported by electronic means such as mobile phones and PDAs and technologies such as RFID, Bluetooth, WLAN, etc. These can be used to automatically record human activities and events in detail. The availability of this contextual information provides an interesting application domain for process mining. The goal of process mining is to discover process models from event logs, i.e. the basic idea of process mining is to identify user processes (behaviours) and extract information about this behaviour by mining event logs for knowledge. In fact, the task of manually constructing templates for complex behaviour is, naturally, a complex task. The developer needs to have a very precise knowledge of what the modelled behaviour consists of, and what it does not consist of. Therefore, this chapter focuses on the potential use of process mining for firstly mining the actual behaviour and secondly measuring behavioural alignment, i.e., comparing the real situation of the user with the expected situation.

7.2 Contextual Situation Recognition

Contextual situation recognition –the task of tracking states and identifying situations– is an important factor to achieve the situation awareness. The purpose of situation recognition is to aid pervasive systems to detect potentially interesting situations. In pervasive environments, context management systems are expected to administrate large volumes of contextual information originating from different domains. Therefore, in order to achieve enhanced situation awareness we need to introduce support capabilities for automatically analyzing and recognizing situations.

To translate the aforementioned Dey's definition of situation, a *fact* is considered as a relation between entities which in turn have properties, and the *situation* is considered as a collection of spatiotemporal facts that are related to each other. An example of a relation could be $isLocatedNear(human_1, human_2)$ which would translate to a fact describing one human is located beside another (where $isLocatedNear$ is defined appropriately). Each of the humans would be entities (possibly having properties) and *isLocatedNear* is the relation which binds them together. As above mentioned, situation recognition can be informally referred to as the task of tracking a specific sequence of states comprising complex arrangement of entities and/or relations in the flow of context information.

A situation assessment is traditionally considered to be a snapshot picture of the system at a given time. This type of situation assessment does not model past states, it does not model the processes that have generated the current state, and it does not allow for prediction into the future. Furthermore, the situations of interest may not be determined from a single snapshot picture, as they are identified by distinct states, which may be separated in time. For example, a busy day situation develops over time and cannot be determined from the system state at a given moment. Finally, the ingredients (e.g. states) used in situation recognition could be originated from different domains the user visits.

Therefore, a situation-aware system has to capture a set of features from distributed context sources and to continuously process these features to derive the overall situation. Thus, major challenges for the creation of situation-aware systems are; to handle the complexity of recognizing a situation, to manage a domain-based sensing infrastructure and to find appropriate reasoning schemes that efficiently derive the overall situation from low-level context features. In the following section a conceptual architecture for the creation of situation-aware systems is presented.

7.2.1 Definitions

We continuously estimate the real-world by using sensor infrastructures. This includes estimating various properties of distinct objects experiencing some behaviour in the environment. In a building or home scenario, we use, for example, temperature, lighting and position sensors to estimate the environment characteristics and people positions. These properties constantly change as people follow their courses of actions. From the user point of view, a process is undertaken which aims at reaching a specific goal, and from the developer point of view a partial part of this process is estimated as it progresses through time.

An *event* describes a change in *state* and a series of these states is called a *history*. A state *s* can be defined as a set of properties describing a process *P* during an interval of time, and an event *E* as a change in state *s* for a process *P* at a specific point in time. As discussed above, when representing situations we can either do it directly in an observable state space, or we can use abstractions on top of this. To this end, we need to resort to relations for describing more complex concepts. A relation could be for example, $isLocatedNear(e_1, e_2)$ which translates to entities e_1 and e_2 being in close proximity (by some definition). A relation can describe relations between an arbitrary numbers of entities; however, this thesis only considers binary relations. Furthermore, when a relation is evaluated and inserted into a system, it becomes a fact $f : r(e_1, e_2)$. Assigning a value to an entity's attribute could be considered as a fact as well: $f : a(e,v)$ where *v* is the value of the attribute *a* of the entity *e*. Inspired by the definitions in [149] process, state and event can be defined as follows:

Definition 1. A process *P* is an abstract model of the user behaviour over a period of time with the aim of achieving a certain objective. Technically, a process is a directed graph of states.

Let $E = \{e_1,...,e_n\}$ a set of entities available in a pervasive environment, and $R = \{r_1,...,r_m\}$ a set of relations between these entities where r_j is a binary relation $r_j(e_p, e_q)$. A fact f_i is either a relation from *R* over a pair of entities e_p, e_q from *E*, or a specified value x_n assigned to an entity's attribute a_l: $f_i : r_j(e_p, e_q) || a_l(e_k, x_n)$. Therefore the state of a process *P* can now be defined as:

Definition 2. A state s is a set of facts $s = \{f_1, ..., f_u\}$ describing a process *P* during an interval of time.

$$s_i = \begin{cases} r_j(e_p, e_q) \forall r_j, e_p, e_q \, (r_i \in R, and \, e_p, e_q \in E) \\ a_l(e_k, x_n) \forall x_n, e_k, a_l \, (x_n \in T(a_l), e_k \in E, and \, a_l \in A(e_k)) \end{cases}$$

Where $T(a_l)$ is the type of the attribute a_l, and $A(e_k)$ is the set of e_k entity attributes.

Definition 3. An event *E* is a change in state s for a process *P* at a specific point in time.

For example, consider a process consisting of three entities e_1, e_2 and e_3. At time *t*, entity e_1 is close to e_2, while e_3 is not close to any of the other entities. The state of the process could be described as a vector $< isLocatedNear(e_1, e_2), isFarFrom(e_1, e_3), isFarFrom(e_2, e_3) >$.

Definition 4. A state sequence *Q* is a vector of states $Q = < s_1, s_2, ..., s_n >$ describing the evolution of a process *P*.

The state sequence definition allows us to capture relations, the state of a process at any particular time, and the changes over time as well. For example, consider Alice in a conference presentation scenario. At the beginning Alice starts presenting, and then Carol asks a question. Meanwhile, Bob enters the conference room. Finally, Alice finishes her presentation and has her coffee. This is illustrated in Table 7.1.

Table 7.1 State Sequence in the Conference Room Scenario

Time	State S_i
1	{Inside(Alice, ConferenceRoom)}
5	{Inside(Alice, ConferenceRoom), Activity(Alice, Presenting)}
15	{Inside(Alice, ConferenceRoom), Activity(Alice, Presenting), Activity(Alice, Talking)}
25	{Inside(Alice, ConferenceRoom), Activity(Alice, Presenting), Activity(Carol, Talking)}
28	{Inside(Alice, ConferenceRoom), Activity(Alice, Presenting),

	Activity(Carol, Talking), Activity(Bob, EnteringConferenceRoom)}
30	{Inside(Alice, ConferenceRoom), Activity(Alice, Presenting), Activity(Carol, Talking)}
32	Inside(Alice, ConferenceRoom), Activity(Alice, Presenting), Activity(Alice, Talking)}
40	Inside(Alice, Lounge), Activity(Alice, HavingCoffee)}

As shown in Table 7.1, there are eight entities (three persons, two locations, and three activities), and two distinct relations (`Inside` and `Activity`). From this state sequence we can extract a number of different situations that could be interesting. For example, one situation could be that Alice is inside the conference room. This situation would cover distinct parts of the state sequence, namely when the relation `Inside(Alice, ConferenceRoom)` exists, which is between time 1 and time 40. Another interesting situation could be that Alice is interrupted by Carol while presenting. This would be another part of the state sequence which in logical notation could be expressed as `Activity(Alice, Interrupted) = Inside(Alice, ConferenceRoom) AND Activity(Carol, Talking) AND Activity(Alice, Presenting)` which is between time 25 and time 32. Another more complex situation, which goes beyond the logical constraints between states, could be identifying if Alice is almost finished her presentation. In this case, she is expected to pass through different states (`Presenting`, `Talking`, and `Interrupted`) in a specified order and repetitions which develops over time. In fact, there is possibly an interesting situation for each possible combination of facts in a state sequence. Therefore, a situation can be defined as follows:

Definition 5. A situation *S* in a state sequence $Q =< s_1, s_2, ..., s_n >$ of a process *P* is a vector of states $S =< s_1^{'}, s_2^{'}, ..., s_m^{'} >$ where each state $s_i^{'}$ in *S* is a subset of a state s_k : i.e. $s_i^{'} \subseteq s_k$ and $s_k \in Q$.

This definition makes it possible to model every potential situation in a state sequence.

7.2.2 Conceptual Architecture

The proposed conceptual architecture consists of five layers: a sensing layer, a facts extraction layer, a reasoning layer, a filtering layer and a situation recognition layer. These layers are depicted in Figure 7.1. Each of these layers is described below.

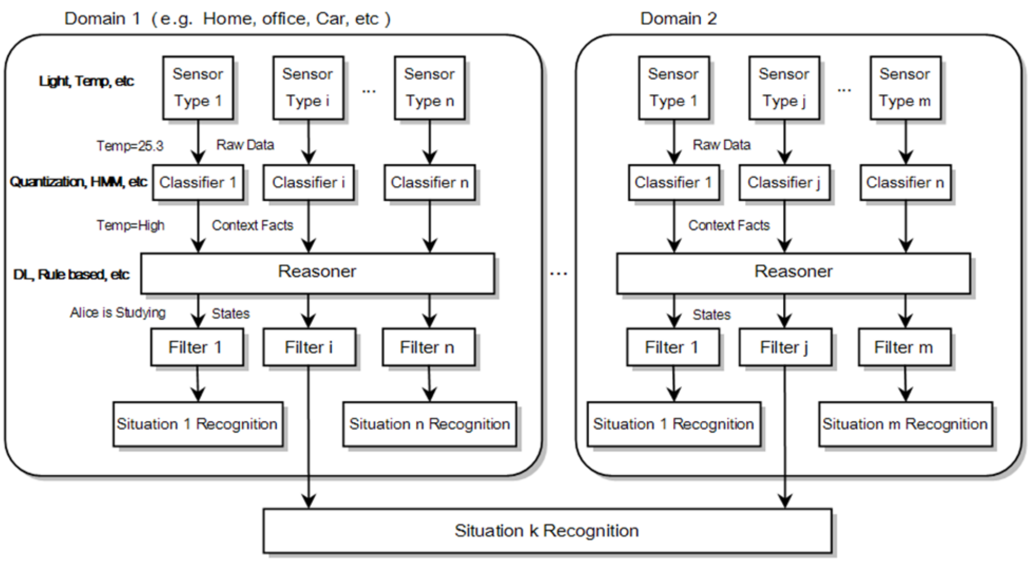


Figure 7.1 Layered Conceptual Architecture

1- Sensing Layer: There is a broad range of different sensors which can be considered for gathering context information like audio, video, a whole wireless-sensor network, etc. These sensors have to be accessed with the help of a specific programming interface provided by the manufacturer of the sensors. So the sensors deliver different types of raw data.

2- Facts Extraction Layer: A classifier is needed in this layer to divide the sensor data into individual classes which are labelled with a symbolic name. Classification could be done simply by a quantization over data or by using more advanced techniques e.g. Rule-Sets, Bayesian-Nets, etc. Therefore, the result of the classifiers is a set of facts which are forwarded to the reasoning steps.

3- Reasoning Layer: Based on the facts resulting from classification, new facts are inferred. This is done with the help of different reasoning schemes, which can be deployed separately or work in parallel. Example reasoning schemes are ontology reasoning, applying description logics reasoners, or rule-based reasoning. The resulting new facts can be further used to identify the different context states. As we will see later, the state is a set of facts.

4- Filtering Layer: In pervasive environments, context-aware systems are expected to manage a large number of contextual facts and states. To facilitate the complexity of recognizing the interesting situations, different situation recognition modules have to be created. Each module is responsible for recognizing a specific situation among the flow of contextual facts. Because each module is concerned with a sub-set of the available states, different filters are needed for different modules. For example, to reason about a meeting situation, we may need to consider only the states: *standing*, *presenting*, and *talking*. In this case, different filtering mechanisms are required to filter out the "noise" states which are not required to reason about a specific situation.

5- Situation Recognition Layer: The main purpose of the situation recognition layer is to recognize the occurrence of an interesting situation among the flow of contextual facts. Because the proposed approach is intended to be generic, the situation recognition module may need to consider different states identified in different domains (areas).

To create situation-aware systems according to the proposed conceptual architecture, the developer can intuitively decompose the relevant situation into different states. Then they can define the different constraints on a sub-set of these states to define the situation as will be seen later. In the following section, the proposed conceptual model that defines the relationships between states, situations, and context elements is described.

7.2.3 Conceptual Model

The concepts of the conceptual metamodel were identified and grouped into two different parts (see Figure 7.2): the context related concepts (white), and the situation related concepts (shaded). As mentioned in [10], the main construct for representing context knowledge is the *Relation* which represents the base context construct that links the context elements: Entity and Attribute.

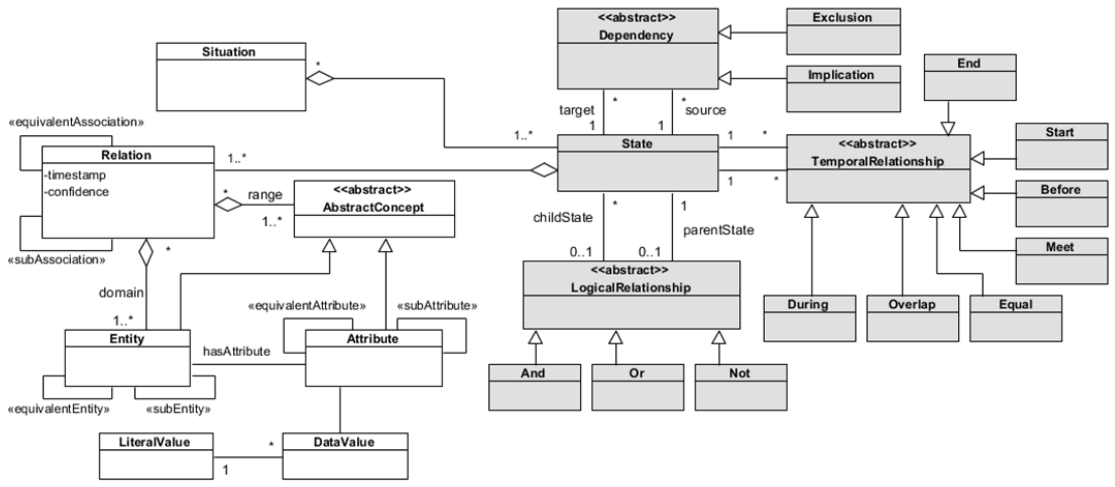


Figure 7.2 Conceptual Model

The above concepts provide the elementary conceptual data pieces to provide the definition of entities, their attributes and relationships between them. However, it is unable to represent complex knowledge, such as the "ready-to-leave-home" situation. To define the situation two concepts are required:

- State class: represents the current state of a specific entity i.e. it is composed of a set of relationships between the entity and other entities as well as a set of attributes values of the entity.

- Situation class: define the situation as a set of states having correlation relationships among them.

The correlations between states can be summarized into three classes:

(1) Dependency: two types of dependencies are identified: *Implication* and *Exclusion*. Implication is used to express the causality between two states. For example, if a person state is *studying* he must have another state: *busy*. *Exclusion* is used to express the conflict between two states. For example, one cannot have a state *cooking* and at the same time he is located in a *LivingRoom*.

(2) Logical Relationship: (e.g. *AND, OR, NOT*) are used in their usual meaning to express different compositions of states: (i) conjunction state (represented by the *AND* class), i.e. when the states s_1 and s_2 are active, then a third state s_3 should be active ($s_3 = s_1 \, AND \, s_2$), (ii) union state (represented by *OR* class), i.e. the state s_3 is active if either s_1 or s_2 is active ($s_3 = s_1 \, OR \, s_2$), and finally (iii) negation state (represented by *NOT* class), which allow to describe that s_3 is active if s_1 is inactive and vice-versa ($s_3 = NOT \, s_1$ and $s_1 = NOT \, s_3$).

(3) Temporal Relationship: A temporal relation is a relation between two states. Since a state is represented by an interval in time, Allen's interval logic [64] is used to handle different possibilities, which are: *before, equal, meets, overlaps, during, starts,* and *finishes*.

7.2.4 Contextual Situation Recognition Algorithm

Having defined the situation, the question now is how to identify situations algorithmically. Two different approaches can be distinguished here: (1) exact matching techniques and (2) approximate matching techniques. In exact techniques (e.g., [59][58]) , all states in a situation need to be found in the context information flow. In approximate techniques, the matching does not need to be exact. Instead, the aim is to determine some degree to which the context information flow matches the expected flow. Approximate techniques require that we establish some form of similarity measures for comparing the extracted context information with the expected one.

Obviously specifying every ingredient in situations that we are interested in finding is a hard task. For example, in the previous scenario, Alice may be interrupted because her presenting laptop has crashed or she may be interrupted by the audience from the beginning before starting her presentation. Furthermore, in pervasive systems we do not often have exact and perfect context information. Thus this thesis focuses on approximate techniques.

To achieve approximate matching two types of constraints can be proposed: (i) constraints (*X*) on the relations between entities and on their attributes values, and (ii) constraints (*Y*) on the temporal ordering of the constraints *X*.

The state sequence $Q = < s_1, s_2, ..., s_n >$ for a process *P* defines the space in which we would like to do the situation recognition. To do approximate matching between the expected situation and observed one we need to have a predefined situation model reference (or template) that imposes certain constraints. A constraint in this model can be defined in three ways. The first option is to define a constraint c_i from a relation in *R* for a pair of entities. The second option is to define a constraint on the value of an entity attribute. Finally, a constraint c_i can be defined from a set of temporal relations $TR = \{tr_1, tr_2, ..., tr_u\}$, where each temporal relation tr_i implies an ordering in time between two constraints $tr_i(c_a, c_b)$. For example, tr_i could be after, before, during, etc.

The Expected Situation Model *ESM* can be defined as *ESM = (Z, C)*, where $Z = \{z_1, z_2, ..., z_s\}$ is a set of variables which during matching will be bound to real entities from *E*, and *C* is a set of constraints $C = \{c_1, c_2, ..., c_h\}$ in which each constrain c_i is defined as:

$c_i = \{r_j(z_p, z_q) \| a_l(z_k, x_n) \| tr_j(c_a, c_b)\}$

The situation recognition problem in the sense of exact matching can be defined as a search for all situations *S* in state sequence *Q*, for which all constraints in an ESM are fulfilled in the situation *S*. However, as we are looking for an approximate matching solution the situation recognition problem could be defined as:

Definition 6 : An approximate solution to a contextual situation recognition problem consists of finding the list of situations *S* in a state sequence *Q* where the value of a similarity function sf between an expected situation model ESM and the situation *S* ($sf = f(ESM, S) \in [0,1]$) is larger than some threshold value.

The following section describes how process mining techniques (e.g. conformance checking) can be leveraged to define the similarity function.

7.3 Process Mining for Pervasive Environments

State log data resulting from the reasoning layer has been used to investigate the applicability of process mining techniques to recognizing situations. In this context, the whole approach can be summarized in three steps:

(1) Self-analysis with the aim to extract the process model. A rich set of detailed entities' state data is recorded over time as a result of the reasoning layer. A systematic and more high-level analysis of these states logs can help to obtain an overall picture of the actual process and understand the user's behaviour. The result of this step is a process model that regroups all potential situations.

(2) Defining the models for the interesting situations. A situation model could be viewed as a sub process model. In this case, the developer has to

manually review the obtained process model and try to identify the possible interesting situations. The result is a situation model called the Expected Situation Model (ESM) which could be viewed as a template model.

(3) Conforming analysis. Having specified the ESM, and based on the observed (recorded) states log, this step considers calculating the degree to which there is a match between the ESM and the observed process model. The following section describes these steps in more detail.

Figure 7.3 gives an overview of the architecture of a traditional process discovery and represents how the proposed approach is integrated to this architecture. The environment consists of context repositories distributed in the environment. The context information regarding users and other entities, their interactions and relations are maintained by the repositories. In the traditional scheme, the context modeller (or the application developer) designs the ESM using her experience and existing approaches (e.g. Petri Net). According to her, this model represents the potential interested situation to track or identify. Then, the model is instantiated and the users or the entities in the environment are expected to follow it during the time, indicated by the grey arrows in Figure 7.3. In this scheme, however, the designed model does not necessarily reflect the actual behaviour the user or entity usually follows as they are not involved in the design of the situation model.

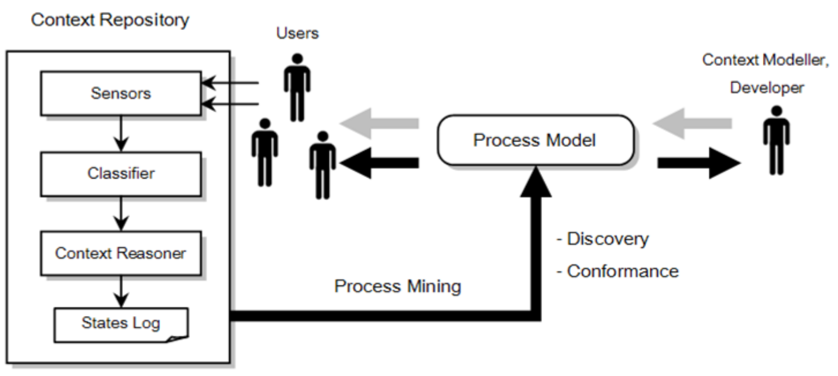


Figure 7.3 Process Mining

The main idea of the process mining is to go in the other direction, as shown by black arrows in Figure 7.3. The states logs correspond to the process instances (particular executions of the process which could be during a certain time interval, in a certain domain, etc.). The process model can be derived from these states logs by using one of the process mining algorithms. Then, the process model can be analyzed by the developer.

In pervasive environments, it is usually difficult to introduce a process model directly from scratch. Using the proposed approach, the existing states logs of several process instances are gathered and automatically generate a model from them. The accuracy of the generated model depends on (i) the number of process instances considered when applying the process mining, and (ii) the chosen process mining algorithm. Obviously the more instances we have, the more accurate the model is.

7.3.1 Abstraction on the States Log Level

These states logs produced by the reasoning layer reflect the different states (or activities) the entities are experiencing in different domains (areas) of the pervasive environment. The focus of this thesis is on control flow mining algorithms which are described at the end of this section.

Sometimes the states logs available in the context repositories contain many details which are not relevant for the process mining algorithms. Thus, we need a technique to abstract from the low level details or even to ignore some state information. This is called abstraction on the states log level. The ProM tool [91] contains a set of filters, which help solve this problem. Table 7.2 shows an example of the recorded states log. In this example, the aim is to understand the user behaviour at home. Thus, here we need to ignore the *Originator* field, filter out the states not corresponding to the *Home* domain, and to map the entities' names to more abstract names. For this purpose, the remap filter can be used to map the entity name to the entity type. Table 7.3 shows the result of this filtering applied to the log of Table 7.2. It shows the corresponding process instances that will be used during the process mining.

Table 7.2 Example of the recorded states log

#	State	Subject	Domain	Timestamp	Originator
1	WokenUp	Alice	Home	2010-01-02T08:23:00.000+01:00	Reasoner1
2	BrushingTeeth	Alice	Home	2010-01-02T08:35:00.000+01:00	Reasoner1
3	WokenUp	Bob	Home	2010-01-02T08:40:00.000+01:00	Reasoner1
4	DrinkingCoffee	Alice	Home	2010-01-02T08:45:00.000+01:00	Reasoner2
5	Shaving	Bob	Home	2010-01-02T08:46:00.000+01:00	Reasoner1
6	Dressing	Alice	Home	2010-01-02T09:00:00.000+01:00	Reasoner1
7	DrivingCar	Alice	Car	2010-01-02T09:15:00.000+01:00	Reasoner3
8	CallingSomeone	Alice	Car	2010-01-02T09:17:00.000+01:00	Reasoner3
9	Working	Alice	Office	2010-01-02T09:30:00.000+01:00	Reasoner4
10	Meeting	Alice	Office	2010-01-02T11:45:00.000+01:00	Reasoner4
…	…	…	…	…	…
32	Asleep	Alice	Home	2010-01-02T22:20:00.000+01:00	Reasoner1
33	WokenUp	Alice	Home	2010-01-03T08:30:00.000+01:00	Reasoner1
…	…	…	…	…	…

Table 7.3 Example of the process instances in Home domain

Alice day1 – (instance 1)	…	Alice Day n – (instance n)	…	Bob Day1– (instance m)	…

#	State	Subject
1	WokenUp	Person
2	BrushingTeeth	Person
3	DrinkingCoffee	Person
4	Dressing	Person

#	State	Subject
1	WokenUp	Person
2	DrinkingCoffee	Person
3	BrushingTeeth	Person
4	MakeUp	Person
5	Dressing	Person

#	State	Subject
1	WokenUp	Person
2	BrushingTeeth	Person
3	Shaving	Person
4	Dressing	Person

Note that during log abstraction different views of the process can be taken, since the definition of what is to be considered a process instance determines the scope of the process to be analyzed. For example, in the context of the conference scenario one could be interested in the overall Alice situations over multiple locations, as well as individual Alice situations within a single room. Thus, different abstractions can be leveraged to obtain different views of the same process.

Since ProM uses the Mining XML (MXML) format to read logs, the states data needs to be converted into this format. The basic structure of MXML is as follows: a process log consists of a set of process instances, which in turn each contains a sequence of events (in our case events correspond to states). A process instance also referred to as case, trace, or audit trail, is one particular realization of the process, while events correspond to concrete steps that are undertaken in the context of that process for the particular process instance. Furthermore, each event carries a timestamp and may contain additional data. An excerpt of a state in such a states log is shown in the following MXML:

```
<Process id="DEFAULT" description="Simulated process">
  <ProcessInstance id="1" description="Simulated process instance">
    <AuditTrailEntry>
      <Data>
         <Attribute name = "subject">Bob</Attribute>
         <Attribute name = "domain">LivingRoom1</Attribute>
      </Data>
      <WorkflowModelElement>DrinkingCoffee</WorkflowModelElement>
      <EventType>started</EventType>
      <Timestamp>2010-01-02T08:23:00.000-07:00</Timestamp>
      <Originator>Reasoner1</Originator>
    </AuditTrailEntry>
    ...
  </ProcessInstance>
  ...
</Process>
```

The state (AuditTrailEntry) was recorded in the context of monitoring Bob’s activities in the first day (Process instance ID is 1) on 2 January 2010 at 08:23:00 according to Pacific Time Zone (Timestamp), and refers to Bob’s domain "LivingRoom1". The dots indicate that the log contains further process instances, and the process instance contains further states.

7.3.2 Control-flow Mining

When dealing with the control flow, the log can be represented as a set of sequences of states (i.e. process instances), see Table 7.3. These instances could be recorded in a time frame, or in a domain. For example, as aforementioned, Table 7.3 shows examples of process instances recorded on different days.

In the process mining area a number of algorithms for control flow mining have been developed, which have different characteristics. The Alpha algorithm [150] can derive a Petri net model from a state log. Another algorithm, the Multi-phase approach [151], creates Event-driven Process Chain (EPC) models from a log, while it first generates a model for each process instance and later aggregates these to a global model. Both the Alpha and the Multi-phase algorithms share the generation and synthesis approach’s precision, i.e. the generated model accurately reflects all ordering relations discovered in the log. While sophisticated filtering of logs can remove noise partially, there are also process mining algorithms which are designed to be more robust in the presence of noise e.g. the heuristics miner [152]. The heuristics miner employs heuristics which, based on the frequency of discovered ordering relations, attempts to discard exceptional behaviour. Because of this feature the heuristics miner has been used in the proposed approach.

Based on the states log depicted in Table 7.3, the heuristics miner has been used to automatically construct a process model shown in Petri Net format in Figure 7.4. It shows the causal dependencies between states and provides an overview about the process model of the user behaviour at home. Having specified the process model, the next step is to identify the interesting situations, i.e. defining the ESMs.

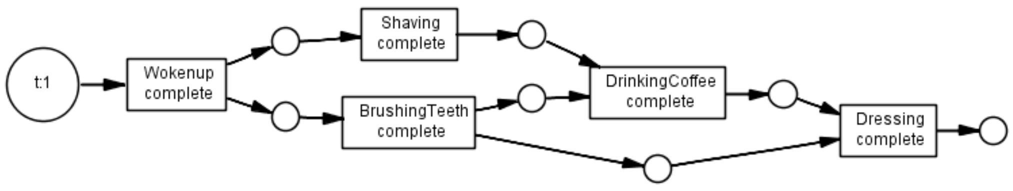


Figure 7.4 Petri Net model of the observed process

7.4 Defining the Expected Situation Model (ESM)

The process model obtained in the first step gives the developer a clear and global idea about the observed behaviour. They can then try to identify and design the model of the interesting situations in which the system can help users fulfilling their tasks. In the previous example, for instance, we may identify the "ready-to-leave-home" situation. The user will be in this situation if he experienced different states e.g. `WokenUp`, `BrushingTeeth`, `DrinkingCoffee`, and `Dressing`. This situation considers the context information available in one domain: `Home`. Recognizing this situation allows the context-aware system to provide users with relevant services e.g. sending a “to do” list to their mobile phone, and switching on the user’s car engine and the air conditioner if necessary.

As mentioned above, the ESM could be viewed as a sub model of the obtained process model. To keep the example as simple as possible and without loss of generality, the ESM is assumed to be identical to the obtained process model.

Recognizing a situation requires a subset of the observed states; therefore, a filter mechanism (Figure 7.1) is needed to filter out the "noise" states when recognizing different situations. Therefore, to recognize the former situation, we need to consider only the domain: `Home` and states whose subjects are of type `Person` and thus we obtain the states log illustrated in Table 7.3.

Another interesting situation which spans different domains could be for example, a "busy-day" situation. This situation considers the different states the user experienced in different domains. For this situation, another filter is implemented that considers all physical domains (e.g. `Home`, `Office`, `Shop`, and `Car`), and the entities of type `Person`.

Having specified the ESM and the corresponding filters, the next step is to match the filtered states log with the corresponding ESM. For this purpose ideas from the process mining and analysis domain have been leveraged.

7.5 Conformance Analysis

Process mining is a helpful tool for context-aware application developers who want to get an overview of how the process is executed i.e. the user behaviour. The question which arises here is how we can determine whether the current observed process is in alignment with our expectation. To answer this question, there exists a set of analysis and verification methods in the process mining domain. One of these techniques is Conformance Checking [153], which takes a log and a process model, e.g. a Petri net, as input. The goal is to analyze the extent to which the observed process execution corresponds to the given process model. In the context of conformance testing this means to measure the “distance” between the behaviour described by the process model and the behaviour actually observed in the log. If the distance is zero, i.e., the observed process exactly matches the ESM specified behaviour, one can say that the log fits the model. Another technique is LTL Checking [56], which analyses the log for compliance with specific constraints, where the latter are specified by means of linear-temporal logic (LTL) formulas. Therefore, the proposed approach considers dual modes of operation: ESM conformance checking mode and LTL constraint checking mode.

7.5.1 ESM Conformance Checking Mode

In this mode, the developer is expected to design the ESM by using the available tools (e.g., Petri nets). This model can be designed in two ways: (i) from scratch i.e. on the basis of the developer experience of designing and understanding the expected behaviour, and (ii) as aforementioned by using one of the process mining algorithms to automatically construct the process model which shows the causal dependencies between states [87][86]. The question that arises now is: does the observed states log conform to the designed ESM? To answer this question, two dimensions of conformances could be distinguished [153]:

- *Fitness*, i.e., the extent to which the states log can be associated with execution paths specified by the process model, and

- *Appropriateness*, i.e., the degree of accuracy in which the process model describes the observed behaviour, combined with the degree of clarity in which it is represented.

This thesis assumes that the ESM is checked and evaluated by measuring some "appropriateness" metrics which is beyond the scope of this thesis. The focus here is on measuring the degree to which the observed behaviour fits with ESM. For this purpose the *Fitness* metric *f* described in [153] has been used. The value of *f* is between 0 (complete mismatch) and 1 (complete match). Measuring the fitness dimension requires recording several instances of the same process and for each newly recorded instance the fitness is re-measured. For this reason the fitness may have the value of 1 if all the instances could be replayed in the model. However, in our case, one instance of the process is considered i.e. the currently observed process. The different states (resulting from the reasoning phase) are continuously recorded thus we need to re-measure the fitness every time a new state is recorded. Obviously, if the currently observed process "deviates" significantly from the ESM the fitness will be low. Moreover, as only one instance is considered, the fitness value may not reach the value 1; therefore, the observed process is considered to matches the ESM if the fitness is greater that a specified threshold which could be estimated experimentally.

To illustrate this, consider the situation of leaving home illustrated in Figure 7.4. Using the metric *f* we can now calculate the fitness between the states logs L_1, L_2, L_3, and the ESM, respectively. Figure 7.5 (a) shows one possible scenario where there is a strong similarity with the ESM model and the fitness measurement yields $f(ESM, L_1) = 1.0$. Note here that this ESM does not consider the order between `Shaving` state and other states; thus; even if the `Shaving` state is observed any time before the `Dressing` state the fitness remains 1.0.

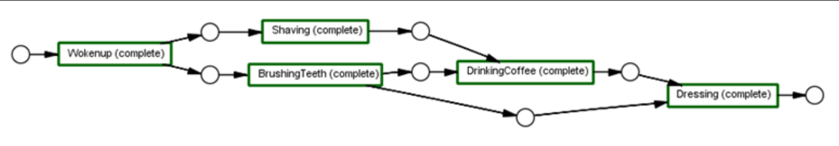
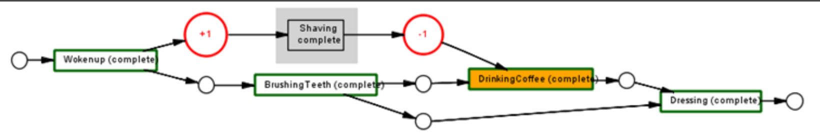
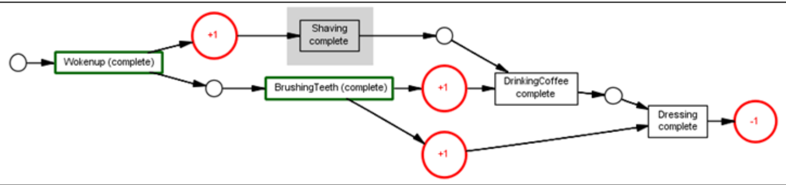
	States Log	Conformance checking	Fitness
(a)	L_1: Wokenup Shaving BrushingTeeth DrinkingCoffee Dressing	 Wokenup (complete) Shaving (complete) BrushingTeeth (complete) DrinkingCoffee (complete) Dressing (complete)	1.0
(b)	L_2: Wokenup BrushingTeeth DrinkingCoffee Dressing	 +1 Shaving complete -1 Wokenup (complete) BrushingTeeth (complete) DrinkingCoffee (complete) Dressing (complete)	0.857
(c)	L_3: Wokenup BrushingTeeth	 +1 Shaving complete Wokenup (complete) BrushingTeeth (complete) +1 DrinkingCoffee complete Dressing complete -1 +1	0.533
⬜ State	🟧 Failed State	🟩 Path Coverage ▨ Remaining state	○ Token Counter

Figure 7.5 Conformance Analysis

Although from a logical point of view L_2 fits well the ESM, replaying the states log L_2 fails since the model requires state `Shaving` being achieved;

the fitness can be measured as $f(ESM, L_2) = 0.857$. As the last log L_3 corresponds to an ongoing process (two states have been achieved so far) the fitness measurement should yield a low value $f(ESM, L_3) = 0.533$. Therefore, if the matching threshold is equal 0.8 for example, then only L_1 and L_2 match the "ready-to-leave-home" situation.

7.5.2 LTL Constraint Checking Mode

In contrast to the model conformance, linear temporal logic (LTL) checking does not assume the existence of a fully defined ESM. Therefore, it can be used to successively introduce, and check for the states succession or dependencies (as described in section 6.2.2 and in the formal definition in section 6.2.4). In this case, the developer can define a set of "rules" using LTL for defining the situation. As an example of the LTL usage in situation definition, Table 7.4 illustrates some LTL expressions and an example for each of them.

Table 7.4 Examples of LTL Analysis

LTL Formula name	LTL Formula Parameter(s) Example	Description
is_last_state_A	A=Dressing	Is the last state equal to A? This formula can be used to define the “ready-to-leave-home” situation without considering the states’ history.

when_A_then_eventually_B	A=Presenting
	B=Interrupted
		If state A occurs, does state B occur after state A occurred? In this example, we may consider that we have a discussion situation.

does_person_P_the_last_state	P=Alice	Is the activity of the last state done by person P? We may consider for this example that if a Person has covered the last state then he achieved a certain situation.

enventually_state_A_and_eventually_B	A=Shaving
	B=BrushingTeeth
		Does state B occur and A too? When measuring the fitness f, we may not be able to know if a certain state has been achieved. In this example, we may conclude that the user is a male.

eventually_state_A_next_B_next_C	A=Dressing
	B=LeavingHome
		Does state C occur after state B occur after state A? In this example, the situation could be

	C=Driving	“heading-to-work”.
always_between_time_T_and_U	T=2010-01-02T09:00:00 U=2010-01-02T10:00:00	Always the timestamp is between T and U. This could be used in combination with other formulas to check if an LTL formula is verified during a certain time interval.

Figure 7.6 illustrates the result of LTL checking if the two logs L_1 and L_2 of Figure 7.5 satisfies the LTL formula `eventually_state_A_and_ eventually_B` where `A=Shaving` and `B=BrushingTeeth`. As can be seen in Figure 7.5 the above formula has been evaluated to false for the log L_2 since the `shaving` activity has not occurred. Whereas the log L_1 states that the activity `shaving` has occurred and followed by the activity `BrushingTeeth` thus rendering the formula result true.

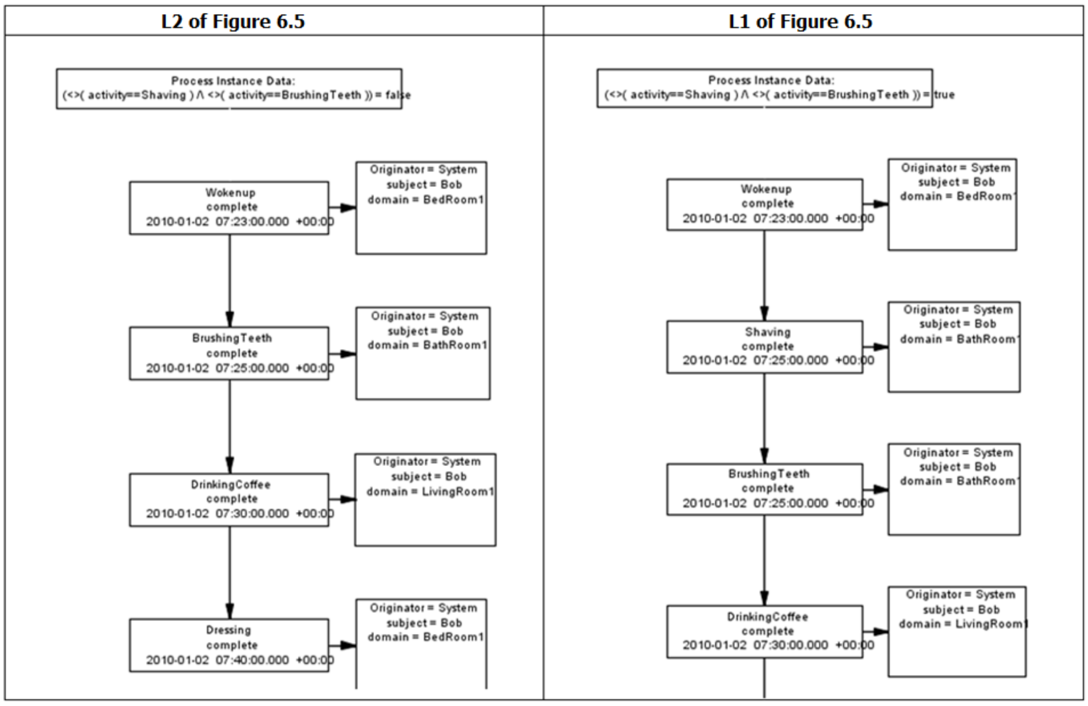

L2 of Figure 6.5
Process Instance Data:
(<>(activity==Shaving) /\ <>(activity==BrushingTeeth)) = false
Wokenup complete 2010-01-02 07:23:00.000 +00:00
Originator = System subject = Bob domain = BedRoom1
BrushingTeeth complete 2010-01-02 07:25:00.000 +00:00
Originator = System subject = Bob domain = BathRoom1
DrinkingCoffee complete 2010-01-02 07:30:00.000 +00:00
Originator = System subject = Bob domain = LivingRoom1
Dressing complete 2010-01-02 07:40:00.000 +00:00
Originator = System subject = Bob domain = BedRoom1
L1 of Figure 6.5
Process Instance Data:
(<>(activity==Shaving) /\ <>(activity==BrushingTeeth)) = true
Wokenup complete 2010-01-02 07:23:00.000 +00:00
Originator = System subject = Bob domain = BedRoom1
Shaving complete 2010-01-02 07:25:00.000 +00:00
Originator = System subject = Bob domain = BathRoom1
BrushingTeeth complete 2010-01-02 07:25:00.000 +00:00
Originator = System subject = Bob domain = BathRoom1
DrinkingCoffee complete 2010-01-02 07:30:00.000 +00:00
Originator = System subject = Bob domain = LivingRoom1

Figure 7.6 Results of LTL checker

7.6 Case Study: Leave-to-Work Situation Recognition

7.6.1 Objective

In this section, a case study has been done to demonstrate that the proposed process-mining based approach is capable of recognizing situations with reasonable accuracy. For this purpose, the following scenario is considered: Alice is a university lecturer. She drives everyday to the university. Alice lives in a cold city; therefore she needs to warm up the car every time she goes to work. In this scenario Alice could be in a "ready-to-leave-home" situation if she experienced different states e.g. `UseToilet`, `PrepareBreakfast` and `TakeShower`. Recognizing this situation allows the context-aware system to provide Alice with relevant services e.g. switching on her car engine and the air conditioner if necessary.

The case study evaluates the approach with the use of a third party smart home dataset, captured in a real-life home environment. The main purpose of this evaluation is to measure the accuracy of the approach for situation recognition.

7.6.2 Background

A fundamental requirement for a pervasive application to be able to act intelligently is the continuous monitoring and understanding of the current situation it is involved in. In addition, the application must recognize situations as they are evolving, that is, in an online fashion. Knowing what is going on is relevant for predicting what will happen, which in turn can be used to make decisions and improve the system performance. To this end, the proposed situation recognition approach does not require the existence of all situation ingredients (states) to perform successfully. Therefore, the aim is to measure the matching between the ESM and an ongoing process that potentially has a situation modelled by this ESM. The goal is to label the situations that an inhabitant is experiencing in a smart environment based on the activity data that is collected by the environment.

To meet these aims, two experiments have been done. In the first experiment, states log describing the different activities performed by an inhabitant during several days is used to mine her daily process and to identify interesting situations. Then, given a labelled set of situation instances, the accuracy of recognizing the situation of interest is measured by measuring the matching between its ESM and their corresponding models. Results show an accuracy of 91.30% for a threshold of 0.75. Secondly, the previous experiments are repeated with situation instances not having all the required states. Results show an accuracy of 73.91% for a threshold of 0.75. These two figures correspond to "leave-to-work" situation as we will see.

7.6.3 Dataset

In this case study a real-life smart home dataset has been used. This dataset contains activities with discernible time durations over a time period. Van Kasteren’s dataset [154] is a public third party dataset that originates from the intelligent autonomous systems group in the University of Amsterdam. It has been widely used by other researchers for smart home experimental evaluations (e.g. [155]). The data is recorded in the home of a 26 year old man over 24 days in his apartment. Annotation was done by the inhabitant via voice recognition from a headset. Over the 24 days, 2120 activities were annotated, resulting in 245 activity instances. Seven different activities were recorded: "leave house", "use toilet", "take shower", "go to bed", "prepare breakfast", "prepare dinner", and "get drink". Only one activity is defined as occurring at any point in time. Fourteen state change digital sensors were installed in doors, kitchen cupboards and kitchen appliances. Each sensor transmits binary values only. A "0" indicates the sensor is not in use, a "1" indicates that the sensor is firing, such as a cupboard sensor indicating that the cupboard is open. Recall that generally speaking we can consider the situation as a specified succession of a set of states corresponding to a set of activities. Part of the activity log of this dataset is as follows:

```
Start time             End time               Activity ID
25-Feb-2008 00:22:46   25-Feb-2008 09:34:12   10
25-Feb-2008 09:37:17   25-Feb-2008 09:38:02   4
25-Feb-2008 09:49:23   25-Feb-2008 09:53:28   13
...
26-Feb-2008 00:39:24   26-Feb-2008 00:39:40   4
26-Feb-2008 03:13:40   26-Feb-2008 03:14:41   4
...
```

7.6.4 Set up and Methodology

Most of the situation recognition research that has been conducted to date focuses on recognizing situations when activities are performed sequentially or when they happen at one point of time. In contrast, the focus here is on recognizing situations in real world when their related activities are omitted and happen in any order. In addition, unlike other works that use the aforementioned dataset for activity recognition, the recognized activities have been used to mine the inhabitant daily process. Knowing this process, different interesting situations could be identified.

As we are interested in mining the daily inhabitant process (behaviour), all the activities in one day are considered as a process instance. By using the heuristics miner, the resulting process model is obtained and shown in Figure 7.7. It shows the causal dependencies between activities and provides an overview about the actual flow of the process, whereas each rectangle corresponds to an activity (the numbers reflect the frequencies at which the activities and their transitions were observed). Note here that as only one activity is defined as occurring at any point in time, and for simplifying process model visualization, Figure 7.7 defines one state type ("complete") for each activity instead of considering two state types "start" and "complete". The ESM of the "leave-to-work" situation, part of the process model, is shown in Figure 7.8.

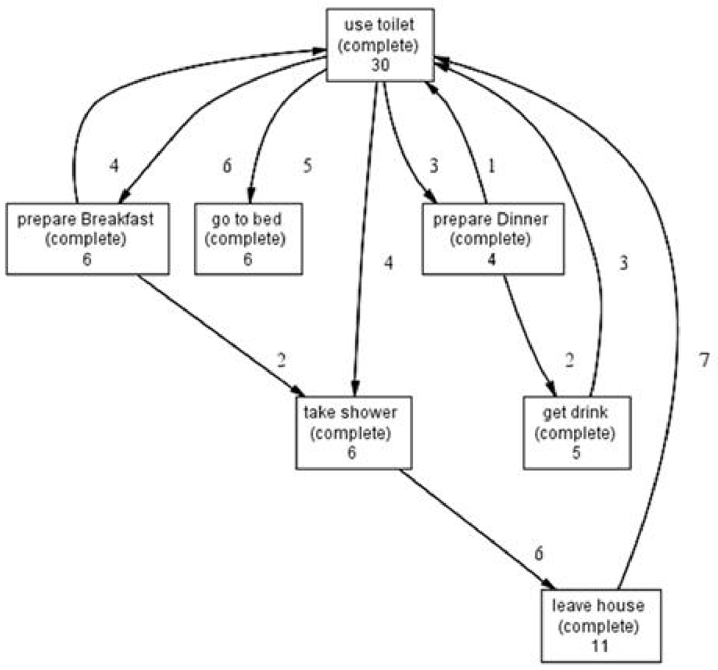


Figure 7.7 The daily inhabitant process model

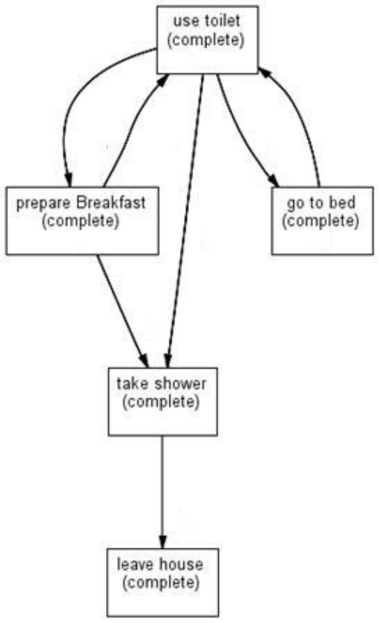


Figure 7.8 The "leave-to-work" situation model

7.6.5 Experiments

To identify the performance of proposed situation recognition, the activity log has been separated into 23 distinct process instances (experiencing leave-to-work situation) corresponding to the activities the user has experienced in 23 days. One-third of these instances have been used to learn the process model shown in Figure 7.7. Then these instances have been used to measure the fitness between each of them and the ESM of the "leave-to-work" situation. Figure 7.9 illustrates the experiment results. It shows that for a threshold of 0.75, 91.30% of the situation cases are recognized. In the second experiment, from each process instance all the activities starting from the activity ID 1 ("leave house") have been dropped, in order to measure the ability of the approach to predict that the inhabitant situation will be most probably that he is leaving to work. Note here that leave-to-work situation is different from "leave house" activity. "Leave house" activity may not be because the user is going to work. Thus, in leave-to-work situation a context history as well as its temporal aspect is considered in the proposed approach. In this respect, the previous experiment has been repeated considering the new process instances and the results are illustrated in Figure 7.10. The figure shows that for a threshold of 0.75, the accuracy of predicting this situation is 73.91%.

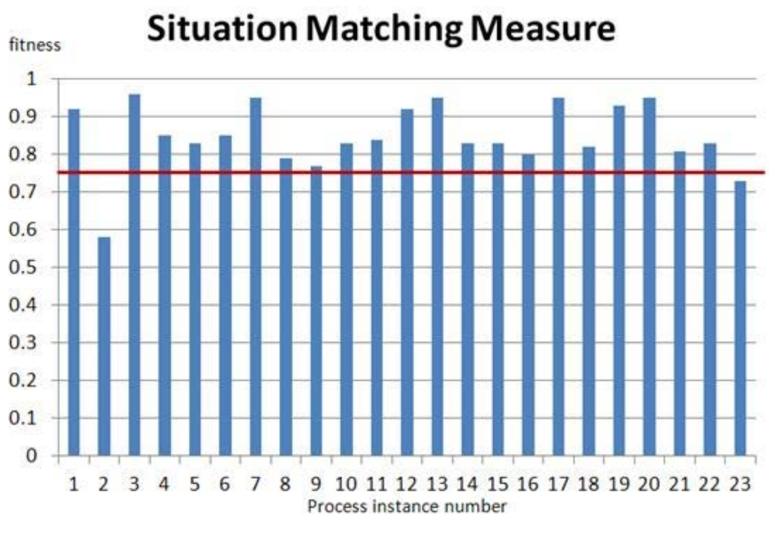


Figure 7.9 Leave-to-work situation matching measure

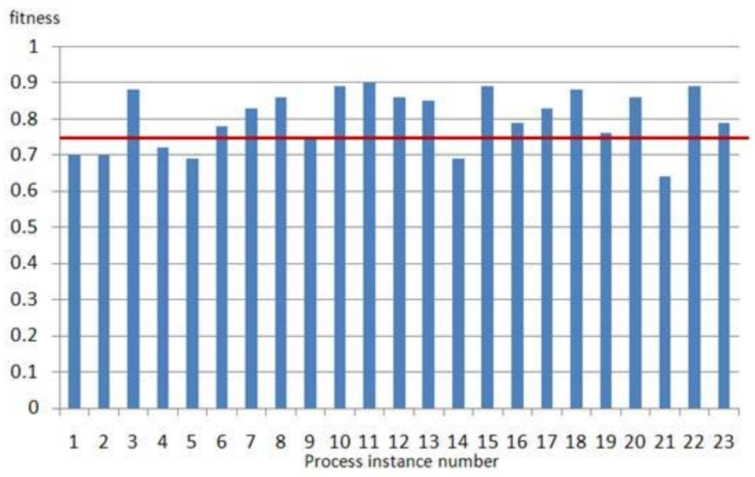


Figure 7.10 Incomplete leave-to-work situation matching measure

7.6.6 Summary

One goal of this thesis is to design an algorithmic approach to recognize situations performed in a real-time, smart pervasive environment. This case study has shown that it is possible to recognize situations that are performed in a smart home and to label an activity stream with high accuracy. Obviously, the accuracy level varied by situation as well as by the threshold considered. This highlights the fact that not only smart environment algorithms are needed to perform automated situation recognition and tracking, but also a reasonable threshold should be determined from experiments on a situation basis.

7.7 Conclusion

Contextual situation recognition is a crucial issue for enhancing the situation awareness of pervasive applications. In this study three essential issues to accomplish situation recognition have been addressed: (1) contextual knowledge gathering - how to gather context knowledge using the conceptual architecture, (2) context knowledge representation - how to represent context data and knowledge concerning situations using the introduced conceptual model, and (3) the algorithm issue - how to track and identify situations. Moreover, it has been argued that approaches that find an approximate matching between an expected behaviour and the observed one are the most suited but require a form of similarity measurement. To this end, it has been shown the potential of structured states log analysis to gain more high-level insight into the user behaviour. Some of the mining algorithms that are included in the ProM framework have been discussed. The extracted process model is then compared with the expected situation model using the conformance and LTL analysis.

Thus, the contribution of this chapter is the introduction of a formalism for the situation recognition problem and the leverage of process mining techniques for measuring situation alignment, i.e., comparing the real situations of users with the expected situations which span one or more domains.

So far the different parts presented in the previous chapters provide the developer with the necessary infrastructure (*ubique* middleware) to acquire the context information of interest for his application. The situation recognition presented in this chapter also enhances the *ubique* middleware to reason about the user behaviour spanning different domains. The next step is to provide the developer with the tools necessary to develop services-oriented applications that takes into consideration the relevant context information to dynamically adapt their behaviour accordingly. Thus, in the next chapter a model-driven approach for service adaptation is proposed.

Chapter 8 *Apto*: A Model Driven Generative Mechanism for Context-aware Adaptive Services

In this chapter, an MDD-based mechanism called *Apto* (the Latin word for adapt) is proposed. It aims at applying an adaptation to services modelled or developed without any adaptation possibility in mind and independently of specific usage contexts. The notion of an *evolution fragment* and *evolution primitive* is introduced to capture the variability in a logical way. Finally, the proposed approach intends to support the viewpoint of context-aware adaptation as a crosscutting concern with respect to the core “business logic” of the service. In this way, the design of the service core can be decoupled from the design of the adaptation logic.

Apto contributes to a solution to automatically generate a customized service based on the current context. Another feature is that *Apto* supplies a set of automated tools for generating and deploying executable service definitions e.g. WS-BPEL (OASIS, 2007) which in turn significantly reduces the development cost.

8.1 The Rationale behind *Apto*

This thesis defines the context-aware service adaptation as the action that modifies the service in a way that causes service behaviour to evolve according to the evolution of business and users’ requirements, and the context considered relevant to that service.

Typically the application developer has to include not only business process in a process (service) language (such as BPEL), but also business rules, policies, constraints, as well as customization mechanisms [156]. Obviously, mixing service with business rules and customization issues weakens the modularity of the system. According to the separation of concern principle, the application developer has to focus on the core application business logic and then define separately the customization and business rules, and weave them to the core application. Therefore, modularization and separation of concerns are the driving principles of the *Apto* approach to target service adaptation.

Further, as the number of services involved in a service-based application grows, the complexity of developing and maintaining these applications also increases. One of the successful approaches to managing this complexity is to represent the application by different architectural views [157]. Examples of these views are orchestration view, control flow view, and component view (see Figure 8.1). This modelling respects the separation of concern principle so that we have multiple views of the system; each view models a specific concern. This chapter focuses on the control flow view; however the proposed approach could be extended to consider the other views.

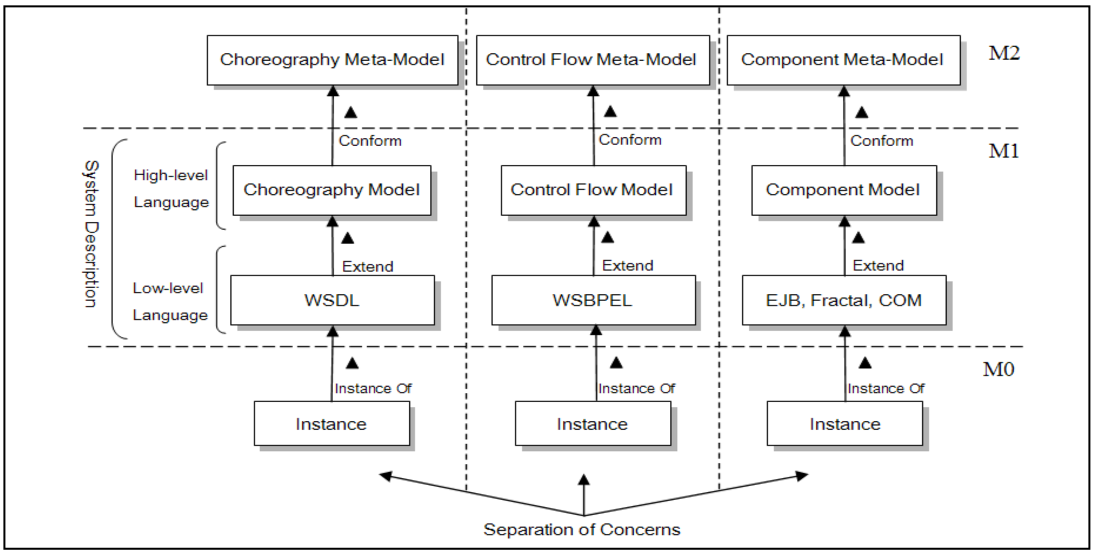


Figure 8.1 Levelled views of service

On the other hand, the process of developing context-aware adaptive services should incorporate facilities to describe the adaptation requirements from early development phases i.e. analysis, design, and implementation to the service execution. This requires languages and modelling approaches that are capable of representing adaptation-specific aspects, i.e., how the service will adapt itself in response to the relevant conditions, events or situations. Therefore, *Apto* adopts MDD methodology. MDD emphasizes using models to capture the application knowledge that is independent of any underlying computing infrastructure (e.g. middleware, programming languages operating systems etc.) which will ease the reuse, adaptation, and evolution of applications.

8.2 *Apto* Approach

The traditional service life cycle, as depicted in Figure 8.2, consists of three phases, namely the design and modelling of the service, the selection or configuration of a particular service variant, and the deployment of this variant in the runtime environment [158]. As the service may evolve over time there should be a feedback loop during which a service is continuously re-adjusted or optimized.

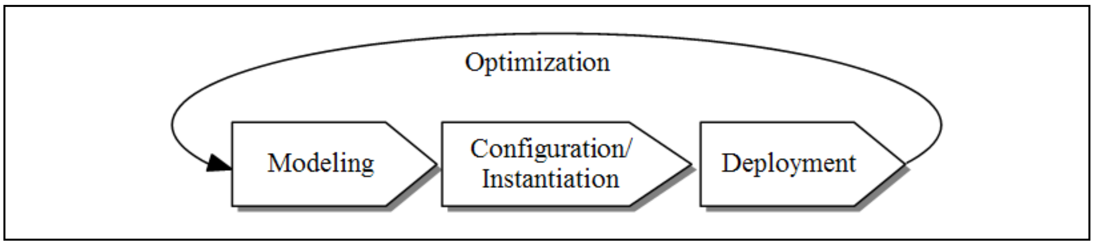


Figure 8.2 Service life cycle

Typically, the developer first focuses on the functional (business logic) aspect of the service which yields a basic model of the service. Then, as will be seen later, they define the evolution fragments and the different possible context scenarios. "Weaving" a group of evolution fragments with the basic model will yield a new service variant. The *Apto* approach is structured in four main sections that address, respectively; the modelling of the control flow, context information, evolution fragments and the linkage model that links between evolution model and context model (Figure 8.3).

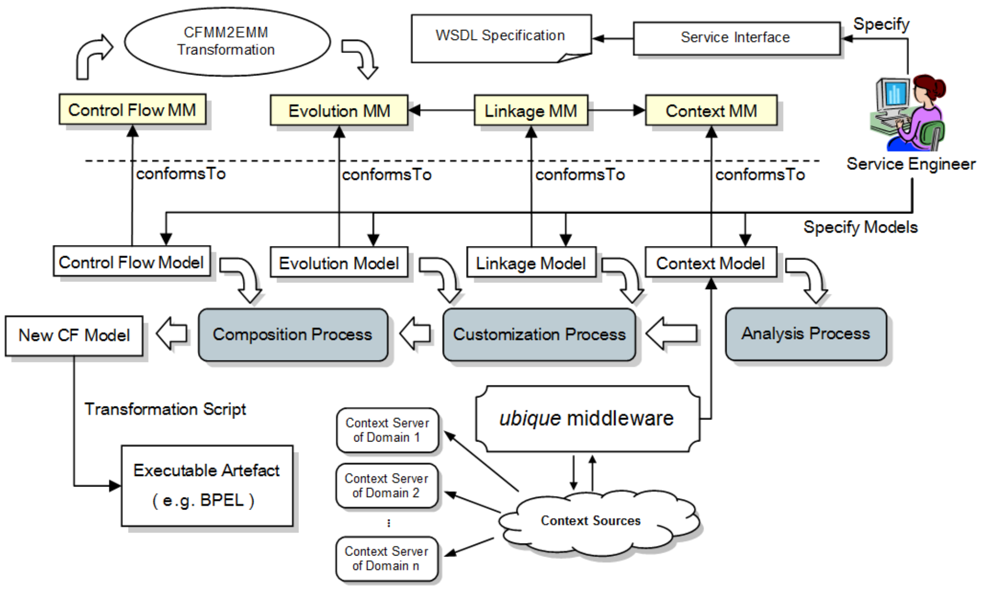


Figure 8.3 *Apto* Approach

During runtime, the user and environmental context will be gathered when the service is invoked by the user. The “Analysis Process” module evaluates all context constraints of the context model. Using the constraints elements evaluated to “true” and the linkage model the “Customisation Process” is able to determine the relevant evolution fragments (see Section 7.3) and the order in which they should be applied to the basic control flow model.

According to the mapping between the evolution fragments and context elements, the set of evolution fragments to be applied to the service could be determined. The “Composition Process” combines these fragments to the control flow model. The result is a new control flow model which corresponds to the current context. All these operations are fulfilled in the model level. Thus, the resulting service model has to be translated to concrete artefacts (e.g. BPEL). It is the role of the infrastructure to create a new instance corresponding to the new control flow model which satisfies the user requirements and context. This transformation from the model to the code is achieved using one of the model-to-text transformation tools.

In the following sections, the conceptual model of the *Apto* approach is introduced; then *Apto* is described in the light of the service development phases: modelling, configuration/instantiation, and deployment.

8.3 A Conceptual Model of Context-aware Adaptive Services

The proposed conceptual model is structured in four main sections that address, respectively, the modelling of the service, context, evolution, and linkage models (see Figure 8.4).

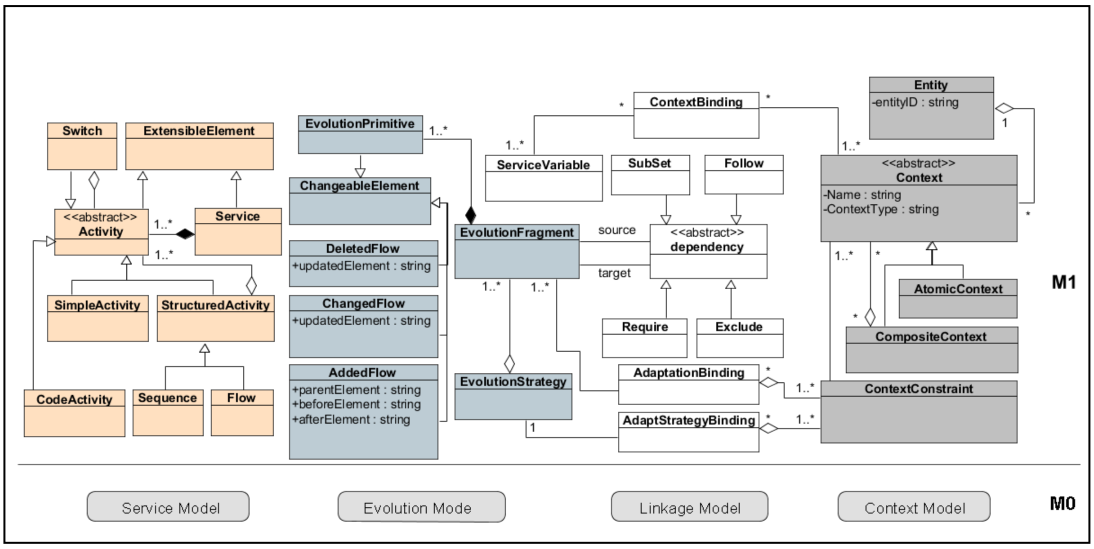


Figure 8.4 *Apto* conceptual model

8.3.1 Basic Service Model

In *Apto* the original service (i.e. an existing service or a newly created one) is denoted as a basic service. The basic service could be defined for the most frequently executed variant of a service family. For illustration purposes, Figure 8.4 depicts some of the main meta-classes representing the key elements of BPEL service model (e.g. Activity, Flow, Sequence, etc.), and their relationships. The service is composed of one or more activities.

8.3.2 Context Model

The main assumption in the proposed model is the representation of relationships between entity and information: entities (such as persons, places, events, etc.) are identified and classified by an ID. Each entity is associated with a set of `contexts` (such as address, location, etc.).

A `Context` is a class that models the context information. The type `Context` is further distinguished into two subtypes `AtomicContext` and `CompositeContext`. Atomic contexts are low-level contexts that do not rely on other contexts and can be provided directly by context sources. In contrast, composite contexts are high-level contexts that may not have direct context source. A composite context aggregates multiple contexts elements, either atomic or composite. For instance, `Temperature` and `RainLikelihood` are atomic contexts provided by e.g. two Web services; whereas, `BadWeather` is a composite context that aggregates these two contexts. The `Name` and `ContextType` properties define the context name and its type and are used in model-to-code generation as will be seen later.

In the context model, a context-dependent constraint concept has been introduced which allows specifying conditions on context elements that must hold to. These constraints correspond to a specified set of evolution fragments that should be applied to the service model in a certain context usage.

8.3.3 Evolution Model

The adaptation in a service usually involves adding, deleting and replacing tasks in the service. In this respect, and in order to achieve a deep change ability, this thesis proposes to add for each class `X` in the BPEL metamodel three classes: `AddedX`, `DeletedX`, and `ChangedX` describing the difference between the basic service model and the respective variant model (see

Figure 8.5). Other change types can be mapped to variations and combinations of these ones. For instance, moving an activity is achieved by deleting the activity and inserting it at a later position of the service.

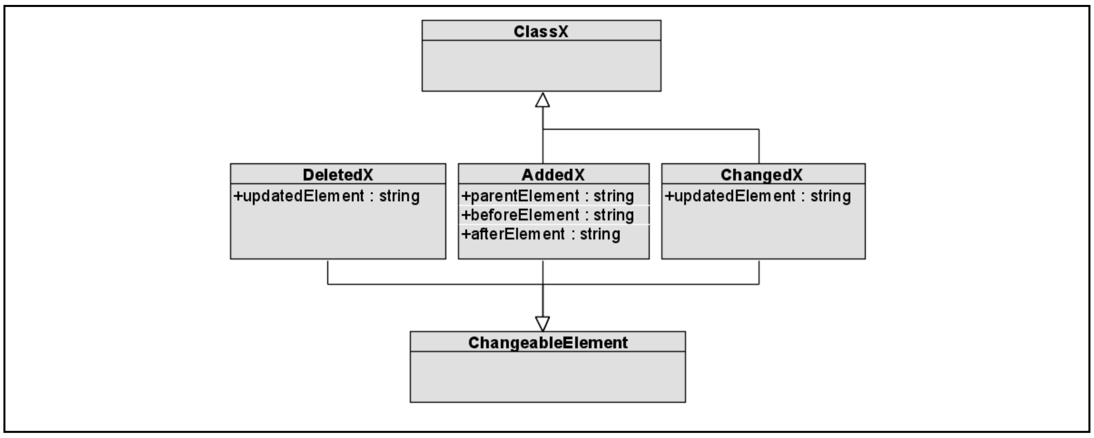


Figure 8.5 Generating evolution metamodel

The evolution metamodel (Figure 8.4) consists of an `EvolutionStrategy` class that contains one or more `EvolutionFragments`. The `EvolutionFragment` in turn consolidates related `EvolutionPrimitives` (a set of elements of type `ChangeableElement`) into a single conceptual variation. The *Apto* approach promotes evolution fragments (EFs) to be first-class entities consisting of closely-related additions, deletions and changes performed on the basic service model.

The evolution metamodel could be automatically generated from the BPEL model. One possible approach is to use the ATL transformation language[6] as in the script of Figure 8.6. Figure 8.4 shows only one example of the three generated classes from the `Flow` class (`AddedFlow`, `DeletedFlow` and `ChangedFlow`).

[6] ATL Language http://www.eclipse.org/m2m/atl/

```
create OUT : EvolutionMM from IN1 : BPELMM, IN2 : MinimalEvolutionMM;
helper def: changeableElement: MinimalEvolutionMM!EClass =
MinimalEvolutionMM!EClass.allInstances()->select(i | i.name = 'ChangeableElement');

rule copyMinimalEvolutionMM {
   from s : MinimalEvolutionMM!EClass
   to t: EvolutionMM!EClass (
        name <- s.name,
        interface <- s.interface,
        eSuperTypes <- s.eSuperTypes,
        eStructuralFeatures <- Sequence {s.eStructuralFeatures}
                ...
   )
}
rule generateEvolutionMMElements {
   from s : BPELMM!EClass (s.name <> 'Service' and not s.abstract)
   to t: EvolutionMM!EClass (
        name <- s.name,
        interface <- s.interface,
        eSuperTypes <- s.eSuperTypes,
        eStructuralFeatures <- Sequence {s.eStructuralFeatures}
                ...
    ),
      added_element: EvolutionMM!EClass (
        name <- 'Added' + s.name,
        eSuperTypes <- Sequence {t, thisModule.changeableElement}
    ),
      changed_element: EvolutionMM!EClass (
        name <- 'Changed' + s.name,
        eSuperTypes <- Sequence {t, thisModule.changeableElement}
    ),
      deleted_element: EvolutionMM!EClass (
        name <- 'Deleted' + s.name,
        eSuperTypes <- thisModule.changeableElement
    )
}
```

Figure 8.6 Evolution metamodel generation script

8.3.4 Linkage Model

Because in the MDD world everything should be a model, the mapping between the context constraints and the EFs is represented by a linkage model. The linkage model is used for three purposes:

(i) Providing context-awareness mechanism. A `ContextBinding` models the automatic binding of contexts to service's input variables. The concept of context binding allows to automatically retrieving available context information. For example, suppose that we have two contexts `ChildrenCount` and `AdultsCount` that represents the number of children and adults using a multi-user service application. These numbers are used by a tourism service to retrieve travel offers from different tourist agents. Thus, a context binding can be built between input parameters of the service and these contexts. The result is that whenever the service is invoked, it will automatically retrieve the number of children and adults and adjust itself accordingly.

(ii) Mapping between the context constraints and the EFs which will be used as information for driving the model transformation. `AdaptationBinding` is actually used as a mapping between a context and an EF. The semantics is that the EFs which have to be applied to the basic service are determined by the value of the context.

(iii) Representing the dependencies between the EFs in order to constrain their use. Each dependency has at least one source EF and exactly one target EF. The relations supported in *Apto* are as follows: dependency (`Require`), compatibility (`Exclude`), execution order constraint (`Follow`), and hierarchy (`SubSet`). `Require` arises when elements introduced by one EF depend on elements introduced by another. The `Exclude` relationship dictates which EFs are incompatible with each another, based on conceptual design knowledge of the service engineer. `SubSet` denotes composition relationship which means that when choosing the child EF the parent EF must be applied first. As one EF might insert an activity whose attributes are changed by a second one, the execution order of these EFs becomes important. Therefore, the `Follow` relationship enables the order in which EFs are applied to the basic service.

8.4 Service Adaptation and Instantiation

The selection of a service variant should take into consideration the service context in which this selection takes place. In addition, this selection should be done automatically. To this end, the basic service model, the defined EFs, the context and the linkage models are used to configure the models of the different variants. A single service variant is created by applying a number of EFs and their related evolution primitives to the basic service.

Step 1. Select EFs: the EFs that are relevant to configuring a particular variant are selected based on the current values of the context model; i.e., an EF will be selected if all context constraints associated with it –via the linkage model– evaluate to “true”.

Step 2. Check EFs relations: EFs relations are considered to ensure service consistency. The selected EFs have to be extended if dependent EFs are missing. Also, it could happen that some of these EFs are mutually exclusive; in this case the service variant cannot be generated. In addition, the EFs are sorted by the order in which they should be applied to the basic service.

Step 3. Apply the EFs: After defining and evaluating the relevant set of EFs, the corresponding evolution primitives are applied to the model of the basic service.

Step 4. Consistency Check: Although the EFs are validated, applying these EFs in combination with each other may result in a deadlock or data inconsistency in the resultant service variant. Therefore, a consistency check is necessary and it is considered in the future work.

Two types of change can be distinguished here: “instance level changes” that should be made on a user request basis and the “permanent changes” that are due to changes of the regulation or the business rules. In the latter case, *Apto* is flexible enough to accommodate this type of evolution by assigning it to a context constraint always evaluated to true. One of the advantages of the *Apto* approach is that the evolution in the service definition can be easily documented.

Further, the evolution fragment concept is used to specify the service adaptation during runtime namely the adaptation strategy. But what about the evolution of the adaptation strategy? Here comes the role of the `AdaptStrategyBinding` concept. An example of the strategy evolution is that the business owner may choose to apply a different adaptation strategy during the Christmas holidays which require them to eliminate, add or change some activities and later to return to the basic strategy. To this end, the evolution strategy could also be linked to a specific context constraint.

8.5 Deployment and Execution

After the adaptation and instantiation, the resultant service variant model has to be transformed into an executable artefact (e.g. specified by BPEL). As the user context as well as the business requirements is in constant change, the evolution and context models should be kept in the runtime as well. This gives the ability to switch between variants during runtime. Obviously for non-long running services the service context is unlikely to change at runtime. However, for long running services, the change in the user or the environmental context may trigger the need to change the service business logic (such as adding or changing activities, variables, or conditional expressions, etc.) i.e. to switch to another service variant. In this respect we can distinguish between two cases:

In the first case, the changes resulting from applying the corresponding EFs to the basic service affect the logic of the service before the current position in the service execution. In this case, the currently running service instance could be considered an obsolete and invalid instance; therefore, a new service variant must be generated and deployed which conforms to the newly operating context. In the second case, changes resulting from applying the EFs affect only the logic of the service after the current position in the service execution. In this case, the instance migration becomes a crucial issue. Recently WebSphere Process Server V7[7] introduced the service instance migration feature that enables service instances to be migrated to a new version of a business service.

[7] http://www.ibm.com/developerworks/websphere/library/techarticles/1008_xie/1008_xie.html?ca=drs-

8.6 *Apto* Tool Realization

As a proof-of-concept, two prototypes have been built to facilitate the proposed approach one on Java platform and the other on .NET platform.

8.6.1 Prototype on Java platform

The Eclipse Modelling Framework (EMF) has been used to model the aforementioned models. Having specified these models, the *Apto* tool is able to deliver the context-aware adaptive service (CAAS) on the basis of the user request as follows (see Figure 8.7). The user's request for the service is intercepted by the Process Proxy service which in turn triggers the Context Analysis module. The Context Analysis module evaluates all context constraints of the context model. Using the constraints elements evaluated to "true" and the linkage model we are able to determine the relevant EFs and the order in which they should be applied to the basic service model. These relevant EFs are used by the Model Composer module which supports context-aware service configuration; i.e., it allows for the configuration of a service variant by applying only those EFs relevant in the service context. The result is the CAAS Model.

This model is automatically transformed, using a set of transformation rules, to generate the executable specification of the target platform. At this time, the proxy service creates a new virtual end point which will be bound to the resulting deployed service. Then it invokes the service deployment of the corresponding execution engine (ODE[8] in the prototype) to deploy the generated service. The user's request is then transferred to the new end point; and the user will be provided with a personalized service that takes into account their context and preferences.

[8] Apache ODE http://ode.apache.org/user-guide.html

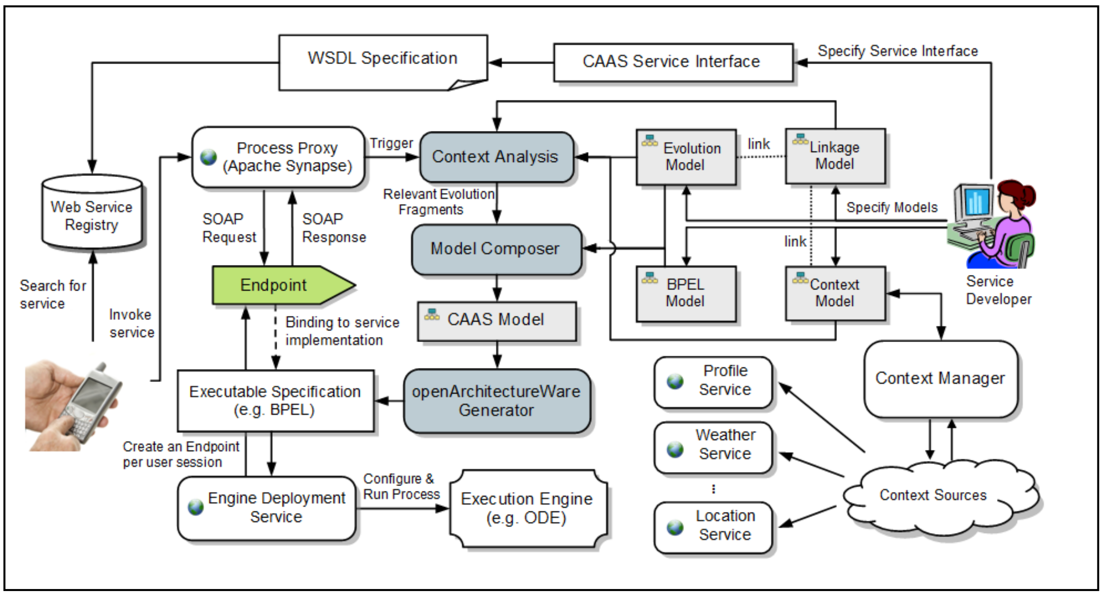


Figure 8.7 *Apto* Java-based tool architecture

For the proxy service, the Apache Synapse[9] has been employed which is designed to be a simple, lightweight and high performance Enterprise Service Bus (ESB). One of the key features of Synapse is that it is easily extended via a custom Java class (mediator); therefore the Synapse engine is configured with a simple XML format to use the proxy service as the mediator. This mediator is responsible for coordinating and running all the above-mentioned modules. The Context Analysis and Model Composer modules are implemented via a Java application. The engine used to run the service is ODE which is an engine for executing services described using the WS-BPEL 2.0 standard. One possible deployment option that is used in the prototype is to deploy ODE as a simple service in Axis 2 (the Apache Web Services/SOAP/WSDL engine) which is invoked using plain SOAP/HTTP and deployed in the Tomcat application server[10].

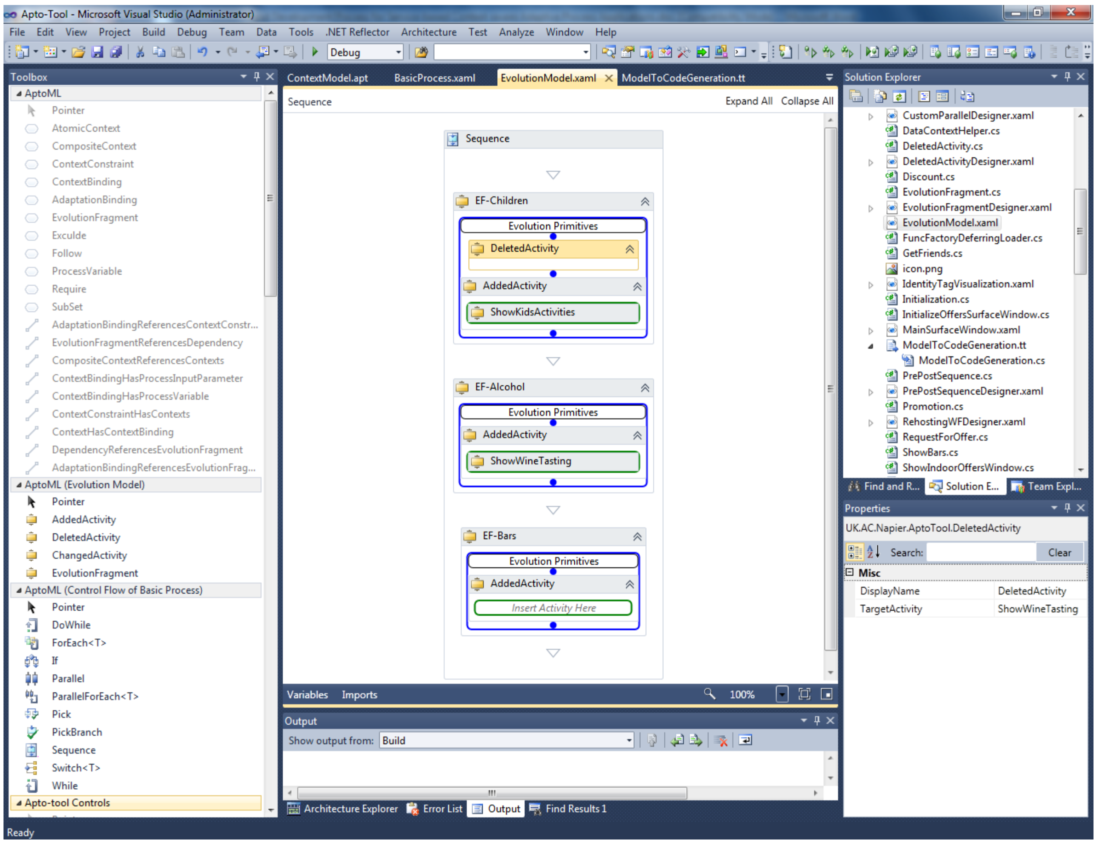
In *Apto*, the model-to-code transformation has been used which takes as input the CAAS model and generates code in an executable language (i.e. BPEL). In the literature there are numerous code generation techniques

[9] Apache Synapse (ESB), http://synapse.apache.org/
[10] Apache Tomcat, http://tomcat.apache.org/

such as templates+filtering, template+metamodel, inline generation, code weaving, etc. [157]. In the *Apto* prototype, the template+metamodel technique has been used which is realized in the openArchitectureWare framework (oAW)[11] to implement the model transformations. But any of above-mentioned techniques can be utilized in the proposed approach with reasonable modifications.

8.6.2 .NET Framework based prototype

A platform has been developed to provide an environment where a service engineer specifies the required contexts and services using high-level and visual modelling languages (see Figure 8.8).



UK.AC.Napier.AptoTool.DeletedActivity

Misc

DisplayName	DeletedActivity
TargetActivity	ShowWineTasting

Figure 8.8 .NET based *Apto* tool

[11] openArchitectureWare, http://www.openarchitectureware.org

The *Apto* modeller (part of *Apto* tool) provides a graphical user interface (GUI) allowing service engineers to specify services using *AptoML* language (see Appendix C). In the implementation, this tool has been developed in C# on top of the .NET framework. A key component of the .NET framework is the Windows Workflow Foundation (WF). WF provides a common framework for building workflows into Microsoft Windows applications. WF itself is a programming model, along with an engine and a set of tools for building workflow-enabled applications. The programming model is made up of exposed APIs that other programming languages can use to interact with the workflow engine. The workflow designer has been leveraged to allow service engineers to design the basic service and the evolution model.

Visual Studio Visualization and Modelling SDK (VMSDK) has been used to create model-based development tools that has been integrated into Visual Studio. VMSDK has been leveraged to the definition of a model that represents *AptoML* concepts. More precisely, it has been used to represent the concepts of context and linkage models. The model has been surrounded with a variety of tools, such as a diagrammatic view, the ability to generate code and other artefacts. This model has been combined with other models (basic service and evolution model) and tools to form the *Apto* toolset.

VMSDK allows us to develop the model in the form of a domain-specific language (DSL). This is achieved by using a specialized editor to define a schema or abstract syntax together with a graphical notation. From this definition we could generate a graphical editor in which users can view and edit the model, serialization methods that save the model in readable XML, and program code and other artefacts using text templating.

In this prototype, service engineers use XAML (Extensible Application Markup Language) [159] to define the service (workflow) and its variants by specifying the evolution fragments. XAML is a markup language for declarative application programming. XAML is used extensively in .NET Framework technologies, particularly WPF (Windows Presentation Framework) and WF. In WPF, XAML is used as a user interface markup language to define UI elements, data binding, eventing, and other features. In WF, workflow definitions can be serialized to XAML. These serialized definitions can be reloaded for editing or inspection, passed to a build system for compilation, or loaded and invoked.

XAML is quite interesting because: (i) it does allow us to model the service from the workflow and UI perspectives in a unified declarative language, (ii) XAML simplifies creating a UI for a .NET Framework application. Visible UI elements can be created in the declarative XAML markup, and thus the UI definition is separated from the run-time logic. Therefore, XAML facilitate the development of services where separate parties can work on the UI and the logic of an application, using potentially different tools. (iii) Standardized by the Organization for the Advancement of Structured Information Standards (OASIS), BPEL is a language for defining system workflows, which is a subset of the more general approach taken by WF [160]. However, the transition from BPEL to WF and vice versa is still possible by using the BPEL Activity Library that implements the constructs defined by version 1.1 of the BPEL specification.

The Context Analysis and Model Composer modules (Figure 8.3) are implemented via a C# application. After weaving the evolution fragments with the basic service model, it loads the new service variant into the workflow engine. It is worth mentioning here that although XAML is used in this case study as a modelling language, the *Apto* approach does not restrict the usage of any specific modelling language; for example, Eclipse Modelling Framework (EMF) can be used instead of XAML (see for example [161]).

8.7 Case Study: Tourism Service Application

8.7.1 Objective

In order to verify and evaluate the *Apto* approach, a case study for a tourism service application running in a multi-touch multi-user table that provides an "intelligent" offering of tourism information is presented here. The aim of this application is to provide users with tailored information and personalized experience when booking for their holidays. Obviously, since usually different users use this type of application simultaneously, considering and resolving conflicts between users' preferences (part of the application context) becomes important. This application runs in a travel agency which has some agreements with other travel agents distributed in different cities. These agents provide a Web service interface for others to get offers and book for their trip. For simplicity the Web services provided by these agents are assumed to have the same interface. In addition, the application displays customized information about the city e.g. historic buildings, art museums, etc. according to the users' preferences.

8.7.2 Solution and Implementation

The basic service model illustrated in Figure 8.9, starts after initialization activity by *ParallelForEach* activity which, given the number of adults and children willing to have a tourist tour, retrieves agents' offers. An activity to show the current discount rate is then launched followed by an activity to show the obtained offers. Next, different tourism information about the city is retrieved and displayed i.e. general city information and available outdoor activities.

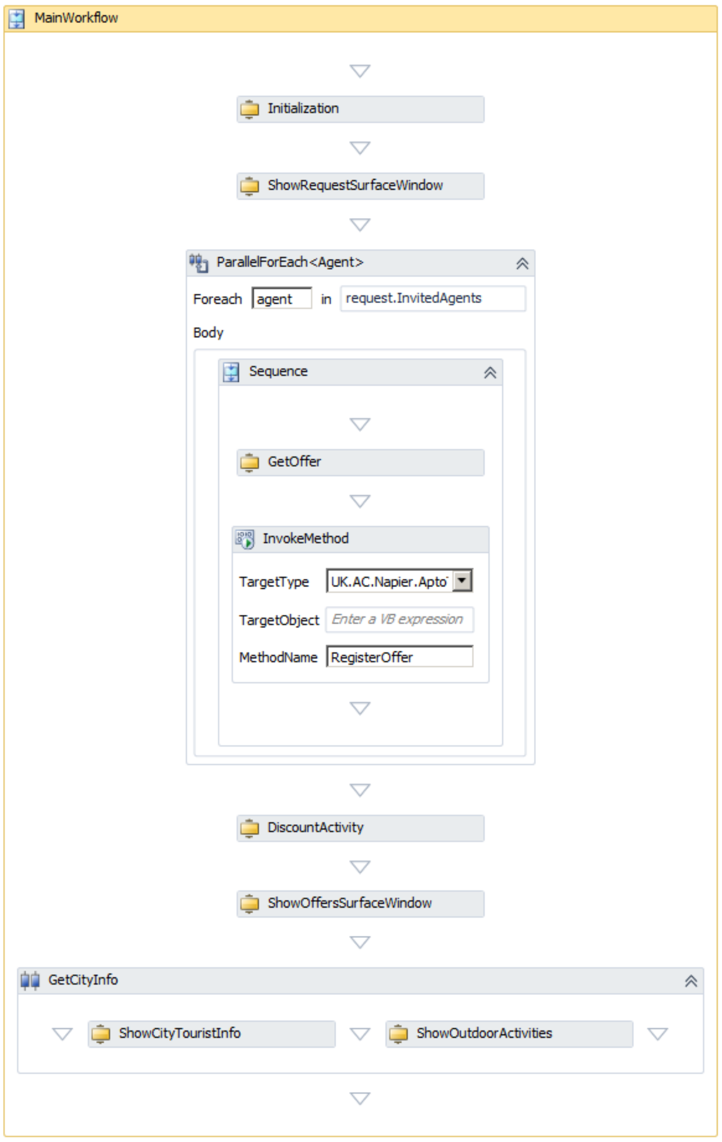


Figure 8.9 Basic service model

The *AptoML* language is used to help service engineers create intuitive service variant models. This section demonstrates: (i) how to specify contexts, (ii) how to define an intelligent tourism service using the *AptoML* language, and (iii) how to automatically transform the service model into executable artefacts.

8.7.2.1 Context-awareness

The specifications of the contexts, including context name and type, are stored in a XAML document, for subsequent usage in the specification of the service. We start by declaring the contexts used in the service, namely *ChildrenCount, AdultsCount, UsersLikeBarsCount, UsersLikeWineCount,* and *ChildrenOriented*. The former four are atomic contexts that are represented by UML classes with the stereotype *AtomicContext* (see Figure 8.10). Figure 8.10 shows the specified context of the tourism service. The atomic contexts *ChildrenCount* and *AdultsCount* are used as input parameters for the service by leveraging the *ContextBinding* mechanism. The *UsersLikeBars* and *UsersLikeWine* context constraints will be evaluated to true if at least one of the users likes wine or bars respectively. During the Christmas holidays the *SalesDay* constraint is evaluated to true. The context constraint *ChildrenInvolved* returns true if the *ChildrenOriented* composite context value is true. The *ChildrenOriented* is a composite context represented by a UML class with the stereotype *CompositeContext*; it is used to determine if the majority of the users are children. The business logic of the aggregation (i.e., how to compute the value of a composite context from its aggregated contexts) is implemented via the *CalculateContextValue* operation of the composite context class.

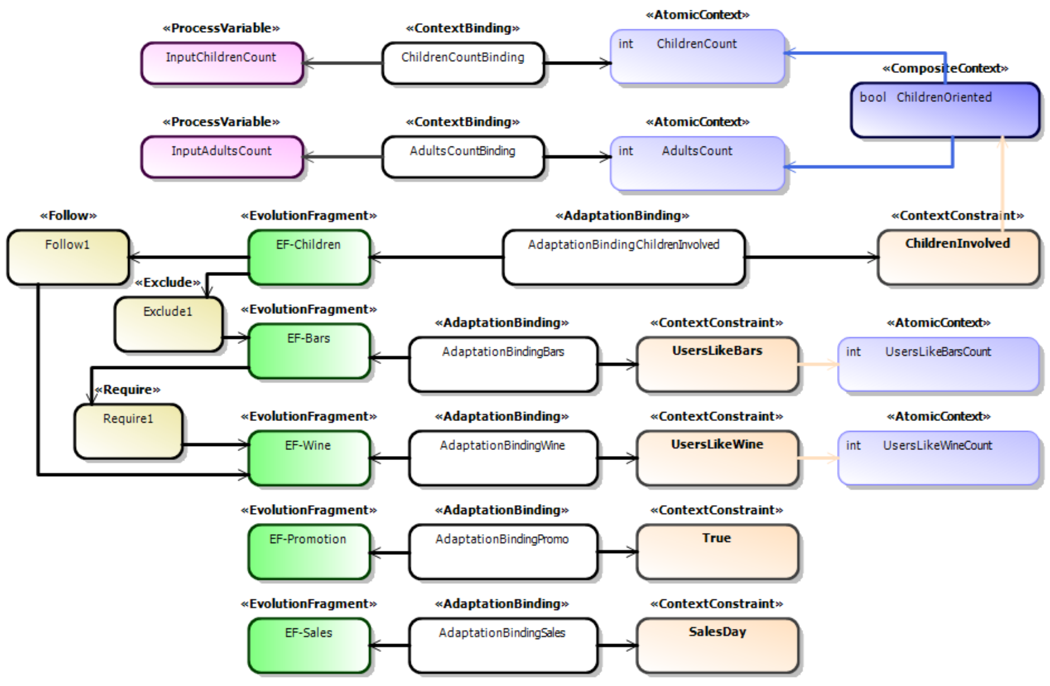
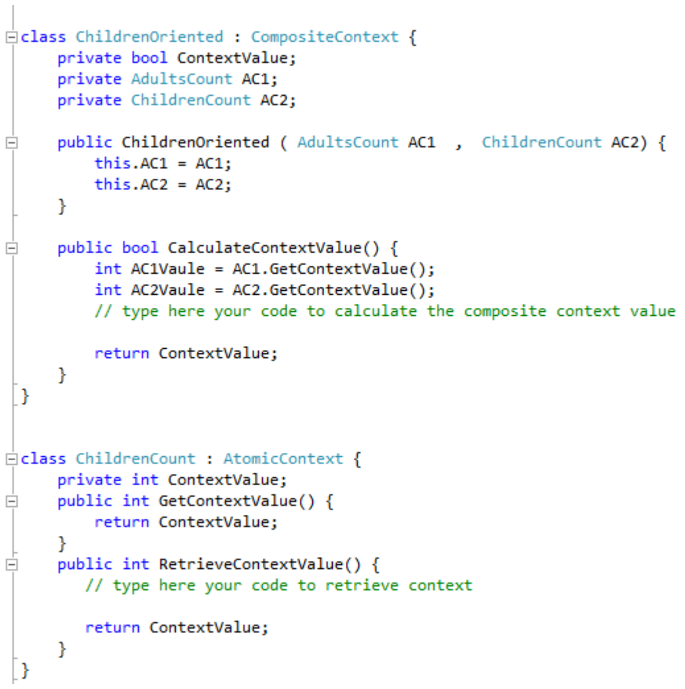


Figure 8.10 Context and linkage models

The *Apto* tool generates a class skeleton so that the service engineer can add the necessary code for the retrieval of the context. For example, Figure 8.11 depicts the generated class for the *ChildrenCount* and *ChildrenOriented* contexts.



```
class ChildrenOriented : CompositeContext {
    private bool ContextValue;
    private AdultsCount AC1;
    private ChildrenCount AC2;

    public ChildrenOriented ( AdultsCount AC1  ,  ChildrenCount AC2) {
        this.AC1 = AC1;
        this.AC2 = AC2;
    }

    public bool CalculateContextValue() {
        int AC1Vaule = AC1.GetContextValue();
        int AC2Vaule = AC2.GetContextValue();
        // type here your code to calculate the composite context value

        return ContextValue;
    }
}

class ChildrenCount : AtomicContext {
    private int ContextValue;
    public int GetContextValue() {
        return ContextValue;
    }
    public int RetrieveContextValue() {
       // type here your code to retrieve context

       return ContextValue;
    }
}
```

Figure 8.11 The generated context class

8.7.2.2 Service Adaptation

After having specified the service context and having designed the basic service model, the service engineer should specify the different evolution fragments. This is illustrated in the following table.

Table 8.1 Evolution fragments and their evolution primitives

Context Constraint	**Evolution Fragment**	**Evolution Primitives**
UsersLikeWine	EF-Wine	- *AddedActivity*: *ShowWineTasting* activity should be added as a child of GetCityInfo.
UsersLikeBars	EF-Bars	- *AddedActivity*: *ShowBars* activity should be added as a child

		of GetCityInfo.
ChildrenInvolved	EF-Children	- *DeletedActivity*: *ShowWineTasting* - *AddedActivity*: *ShowKidsActivities* activity should be added as a child of *GetCityInfo*.
SalesDay	EF-Sales	- *ChangedActivity*: *Discount* activity should be changed to reflect the new discount value.
True	EF-Promotion	- *AddedActivity*: Promotion activity should be added after the GetCityInfo activity.

The "True" constraint means that this is a permanent change that should be applied to the basic service model. Table 8.2 shows the different dependencies between the specified evolution fragments. The dependency 1 means that the *EF-Children* should be applied after applying the *EF-Wine* i.e. the *ShowWineTasting* should be dropped and then the *ShowKidsActivities* should be added. If users like going to the bars then they presumably like wine. This is expressed by the dependency 2 which means that if one user likes going to the bars two activities will be added *ShowBars* and *ShowWineTasting*. Finally, since the *EF-Children* and *EF-Bars* are assumed to be mutually exclusive, the generation of a service variant is not possible if these fragments should be applied simultaneously (dependency 3).

Table 8.2 Evolution fragments dependency

#	**Dependency**
1	EF-Children follow EF-Wine
2	EF-Bars require EF-Wine
3	EF-Children exclude EF-Bars

Figure 8.12 depicts a part of the service model after applying *EF-Children*, *EF-Sales*, and *EF-Promotion* evolution fragments to the basic service model.

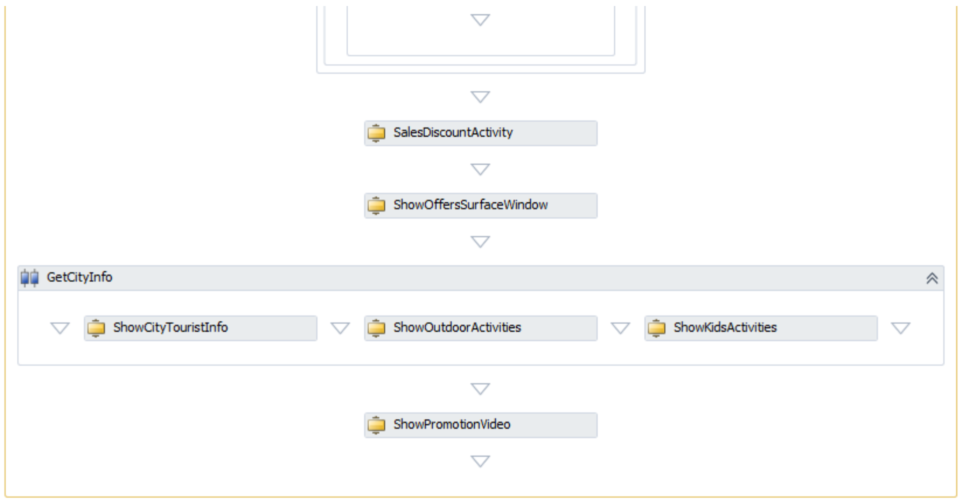


Figure 8.12 Example of a service variant model

8.7.2.3 Transforming Service into Executable Artefacts

After having defined the service using the *Apto* tool, the model transformer comes into play during the model transformation process. This process takes as input the XAML document of the service model -produced by the *Apto* tool- and applies the relevant EFs according to the retrieved context values to derive the correspondent service variant. Then it converts the service model into executable Web service specifications (i.e. BPEL and the relevant configuration files). However, since the .NET workflow engine is used in this case study to execute the service this step is omitted.

8.7.3 Summary

This case study has illustrated the model driven approach for the development and evolution of context-aware services, realised by *Apto* platform. The approach is also supported by *AptoML* language and tools conceived to ease and to increase the automation in developing and evolving of such services as well as decoupling service development from context handling layer.

The case study shows that using *Apto* tool the developer is able to logically see the service adaptation as deriving a service variant by applying to a basic service a set of EFs corresponding to different usage contexts leaving the resolution of the EFs dependencies to the implemented tool. Resolving the conflicts and dependencies between EFs is simple in this case study; however, it could require a rule-based system or modelling the EFs in a semantic language for more complex scenarios.

The generative aspect of the approach (e.g. Figure 8.11) saves the developer time as they focus, with the help of the tool, on the business logic of the service and its variants in different contexts leaving the task of acquiring the relevant context, and choosing and instantiating the service variant to the implemented tool.

The result shows that the approach and its supporting platform are effective for the problem and promising for real-life applications.

8.8 Conclusion

Change is the only constant in the software/service development world due to the evolution of business or user requirements. Therefore, there is a need to customize services by generating a service variant that corresponds to the change in the business and user requirements. The *Apto* model-driven approach for managing and generating service variants has been described.

The novelty in this chapter lies in (i) the introduction of the concepts of evolution fragments and evolution primitive to enable the developer to logically view the service variant i.e. in terms of the features that determine the difference between service variants in each usage context, (ii) and the generative aspect of the approach to automatically derive the service variant corresponding to the available context.

One of the advantages of using MDD is that the context management and adaptation logic are included in models rather than directly implemented in code. Based on logically-viewed well-defined evolution fragments and evolution primitive constructs; on the ability to group evolution fragments in components; and on the ability to regroup these components in a constrained way, necessary adjustments of the basic service can be correctly and easily realized when creating or configuring a service variant.

Finally, *Apto* adopts the viewpoint that this kind of adaptation can often be considered as a crosscutting concern with respect to the core service logic. Hence, one of the *Apto*’s main goals has been the decoupling of the design and implementation of the adaptation logic from the design and implementation of the main service logic.

Chapter 9 Conclusions and Future Work

The main outcome of the research undertaken for this thesis was the development of a new approach for the development and evolution of context-aware services which regroups four main parts: a new context modelling approach, a cross-domain context management middleware, a contextual situation recognition algorithm, as well as a mechanism for generating context-aware adaptive services. To achieve these objectives different techniques have been leveraged such as the software product line, model driven development, process mining, and the Jabber protocol.

This chapter discusses three parts of the work that merit further examination and discussion. Firstly, the evaluation of the proposed approach is carried out in terms of their strong and weak points. Secondly, the conclusions are reached and the main contributions are summarised. Thirdly, the future directions of the research are discussed.

9.1 Critical Analysis

9.1.1 Context as a Dynamic Product Line

The proposed context modelling approach can be seen from two perspectives: (i) identifying context features and giving them semantics by mapping context feature models to OCM; and (ii) using feature models to provide a representation of variability in context models. The proposed meets the requirements mentioned in Section 3.1.1:

R1- Efficient applicability of context reasoning: unlike the reasoning on monolithic context information proposed for example in CoBrA [31] and CONON [33], the proposed approach provide a mechanism (i.e. context feature model configuration) for pre-selection of context information relevant to an application. This could speed up the reasoning process by reducing the size of the knowledge base.

R2- Ease of context querying: in the existing approaches (such as CoOL [98], CONON [33], GAS [99], and CoDAMoS [100]) queries are defined in general-purpose querying mechanisms (e.g. SPARQL) or a domain-specific query language (such as [44]). On the other hand, Gaia [1] introduced the *context file system* to construct a virtual directory hierarchy, based on the types of context associated with particular files. This virtual directory hierarchy forms a simple query language to determine what types of context are attached to files. On the other hand, the proposed context feature model allows the context modeller to devise context-specific features that can be shared among all applications. Moreover, retrieving context information using general-purpose query mechanisms remains possible by devising a special context feature.

R3- Providing different levels of abstractions: As aforementioned in section 3.1.3, the only approach that provide the developers with mechanisms to specify the abstraction level of the context information of interest is the context model of ACAI [113]. In ACAI the highest level of abstraction is the *ContextView* which represents the different types of context that belong to a given entity. Thus *ContextView* represents the primary dimension which will be used to access the secondary context. *ContextView* has two properties *contains*, and *invokes*. The classes *ContextFeatures* and *ContextEngagements* are the respective ranges of those properties. These classes are considered to be the second level of expressiveness in the ontology. However, ACAI ontology is rather elementary with regards to the context features and types defined. In addition, the proposed approach in this thesis is more generic since it does not impose any restriction either on the number of these dependencies or on the number of context features. Finally, it gives the possibility to provide the context information on arbitrary levels of abstraction thanks to the arbitrary composition of context primitives e.g. inference rules.

R4- Efficient context provisioning: In order for the applications to access the relevant context information, most of the existing approaches present suitable access paths in the context modelling. For example, by using the context file system of Gaia, primary dimensions which will be used to access the secondary context can be easily identified and mapped to file paths. However, from the context modeller usability perspective, the proposed approach in this thesis is more intuitive and it allows the modeller to think about the context information from different perspectives. Thus he is able to use the feature model available tools to design different context feature models corresponding to context access paths.

R5- Provide constructs to model context variability: None of surveyed modelling approaches provide constructs to model context variability. The only exception is the CoOL model which partially supports context variability by introducing the aspect-scale model. However, it is not generic enough to model the aspects hierarchy and their dependency. In addition, CoOL is less practical for expressing aspects' scales with regards to more non-material context data, such as user preference or activity. On the other hand, the feature-based context modelling uses the context features, their relationships and dependencies to provide more generic solution to address the context variability. In addition, the use of context-specific features would improve the overall performance of the system, since it might decrease the number of network interactions between an application and the context provider. Finally, it might reduce the reasoning time by considering only the relevant context primitives.

However, although the proposed approach provides the application with a customized view of the available context information after receiving its queries, it is the application's responsibility to further query this acquired information using, for example, the SPARQL language. In addition, any change in the available context information triggers the context information generation process and thus delivering the newly available context information to the application which may not be efficient in terms of the network bandwidth usage. Thus there is a need for extending the work to consider delivering only the context information corresponding to the context features affected.

Finally, in its current implementation the proposed approach does not reflect the dependencies between the different context features in a semantic way. Thus, there is a need to extend the proposed approach to model these dependencies using a semantic language.

9.1.2 *ubique* Middleware

In this section, the *ubique* approach is analyzed with respect to the requirements set out in section 3.2.1:

Domains of context perception: This requirement, which is compliant with the principle of system boundary of pervasive applications, is achieved by using CS in each domain and the dissemination between CSs across different domains. Classical work in context-aware computing has developed centralized and application-specific solutions such as Context Toolkit [32] and Gaia [1]. These approaches offer solutions for restricted and small-size smart space environments, with localized scalability. More recent middleware offer access to context information in distributed repositories e.g. Confab [4], PACE [120], CAMUS [5], and GLOSS [6]. However, unlike these approaches, the notion of home domain CS in the *ubique* approach simplifies application developments as it is the reference point for any context information related to the entities registered in it.

Uniform API interface and protocol: By providing the *ubique*'s set of open and generic APIs, context is made available to third party application developers to build new services without having to define specific mechanisms for context distribution and management between domains. In addition, these APIs and the proposed protocol between different entities enable external providers and consumers to be integrated into the *ubique* system to provide or consume context information.

Efficient context information dissemination: Since communication resources are limited, and since most context information gathered by a context server will not necessarily be used by any application, *ubique* considers filtering and disseminating only the context information that is explicitly required by an application.

Cross-domain reasoning: Usually, when the user roams between different domains his context information is stored in the context repositories available in the visited domains. In this scenario, recognizing contextual situations spanning more than one domain may be a difficult task for the developer using the exiting context management middleware. This is because the developer has to identify and resister the required queries in the context repositories holding the context information of interest (e.g. PACE [120] and Confab [4]). Then he has to use the acquired context information from different domains to recognize contextual situations. Unlike the existing approaches, *ubique* provides an enabling infrastructure to support reasoning about the context information across different domains and to identify the contextual situations which span different domains. Moreover, this enforces the idea that each domain should have its own inference mechanism whereas in the HDS a cross-domain inference becomes possible.

Dynamic matching between context providers and consumers: In *ubique* the matching function of the context manager ensures efficient context information dissemination. In addition, since the CPs specify their capabilities in providing context information that correspond to different domains, an application can specify in its interests or queries the domain(s) from which it is interested in retrieving the context information.

Support for privacy: The most representative example of approaches addressing the privacy issue in the context management is the Context Fabric (Confab) [4]. It provides architecture for privacy-sensitive systems, as well as a set of privacy mechanisms that can be used by application developers. However, the user has to define and submit a separate privacy policy for each domain. In addition, as the context information is distributed in different repositories enforcing the user’s privacy may not be an easy task. On the other hand, in *ubique* approach, since the context information is centralized in one CS (HDS), enforcing the user’s privacy policy which spans different domains is feasible. In addition, the dissemination protocol between CPs and CSs on one hand, and between CSs on the other hand, ensures that the context information will not be stored everywhere and that this information will be disseminated only if the receiver has the privilege to get it.

However, *ubique* still has some limitations. For example in its current implementation only ID-based queries are supported which are answered by the HDS; however, queries like "give me all users in the train 123 that are reading" would also need to be answered by the respective domain server. In this case the privacy argument may not be sufficient as it is only valid when we have ID-based access.

In addition, in the *ubique* approach the concepts of ownership and ID are closely linked but in reality they should be considered as separate aspects. For example, although Alice could be always automatically considered as the owner of information pertaining to her, it is not the case for the object entities. Further, additional evaluation of the *ubique* approach is needed in terms of its scalability and its applicability in more complex real-life scenarios. In terms of the dissemination latency of contextlets among servers it is probably not quite representative as the latency is dominated by actual cross-domain network bandwidth and the middleware implementation optimization.

Finally, although the definition of the domain has been already established in the literature, additional questions remain to be answered such as: how does the user find the domain he is currently in? What kind of infrastructure is needed to support that? Can there be overlapping domains?

9.1.3 Situation Recognition Approach

One goal of this work is to design an algorithmic approach to recognize situations performed in real-time and in a smart pervasive environment. Three features for recognizing situations in pervasive computing have been identified: the use of context history, the use of context in different domains, and approximate matching. Based on these features, a process mining based approach has been proposed to derive the process model of the user activities from recorded state logs.

The existing rule-based (e.g. [10][11]) and ontology-based (e.g. [33]) approaches provide the flexibility to represent a situation in multiple ways. In addition, the modularity of representing situations emphasizes the incremental approach and reuse when building a knowledge base of situations. However, in the domain of context-aware computing, these approaches are error-prone due to the incompleteness and ambiguity of context information. In addition, they use exact matching techniques (e.g. [59][58]), where all states in a situation need to be found in the context information flow. Thus they are not suitable for inference from imprecise and incomplete contexts as they are designed for exact reasoning. The proposed approach in this thesis follows an approximate technique where the matching does not need to be exact. Instead, the aim is to determine some degree to which the context information flow matches the expected flow.

The machine learning techniques (e.g. [60]), on the other hand, have at their core a probabilistic reasoning method to, in the first instance, learn behaviour patterns and follow this to recognise situations. Typically the existing approaches to situation detection require constructing sequence-based models of low-level activity features. However, this thesis argues that activities may have a distinct series of activities but with no particular sequence. Therefore, the proposed approach in this thesis uses process mining techniques to situation recognition by relying on the relevance weights of activities rather than on sequence information.

Some of the existing approaches (e.g. [60]) rely on the training data to learn the behavioural patterns which requires large amounts of activity historical data which can be difficult and costly to acquire. Other approach reduces the reliance on training data by incorporating domain knowledge into their approaches (e.g. [15][128][129][130]).

On the other hand, the proposed approach in this thesis has the advantage of allowing context modellers to create models (from scratch or inspired by the derived process model) for user’s situations which take into consideration the different activities the user may experience in the different domains they visit. For this purpose, context modellers have to design for each situation the necessary filters to filter out the “noise” activities which are not related to the recognition of the situation in question. The recognition is conducted in the conformance testing technique that evaluates if the current observed state is in alignment with the created (expected) model.

The experimental results indicate that it is possible to recognize situations that are performed in a smart home and to label an activity stream with high accuracy. As aforementioned in section 2.3.2, the process mining can deal with various forms of concurrency. Additionally, the accuracy level varied, obviously, by situation as well as by the threshold considered. This highlights the fact that it is not only smart environment algorithms that are needed to perform automated situation recognition and tracking, but also a reasonable threshold should be determined from experiments on a situation basis.

The evaluation of the approach uses the activity data in a widely-used data set, and infers the "leave-to-work" situation. However, the evaluation should be extended to include not only other types of situations but also it should consider using the whole architecture layers. Currently the case study uses the activity data but does not consider using the sensor data to identify these activities. In addition, similar to the learning based techniques, deriving the user process model requires large amounts of activity historical data which can be difficult and costly to acquire. However, the contextual situation model could be build from scratch based on the domain expert knowledge.

9.1.4 *Apto* Approach

Unlike the existing approaches which address the adaptability in the code level (e.g. eFlow [133], Context Oriented Programming [139], AO4BPEL [132], and VxBPEL [20]), *Apto* presents a model-driven approach to support the adaptation of the service.

Some approaches (e.g. [20][82][137]) incorporate variabilities into the service model. They tackle the service adaptation on the service instance or definition level by explicitly specifying some form of variations (i.e. variation points and variants) in the service model that will be determined at design time or runtime according to the operating context. For example, in VxBPEL [20] the variation points and variants are embedded in the service logic itself which weakens the system modularity and violates the separation of concern principle. In addition, the constructs used to specify the service variant (i.e. variation points and variants) do not reflect the way the developer or designer logically view the difference in the service model in each context usage. On the other hand, *Apto* captures the service variability in a logical and intuitive way by introducing the notions of *evolution fragment* and *evolution primitive*.

Further, unlike some approaches such as AdaptiveBPEL [131] AO4BPEL [132], which requires modifying the BPEL engine, the *Apto* approach generates the adaptive service artefacts in the standard BPEL language and does not require the extension or modification of the BPEL engine. In addition, the proposed mechanism could apply an adaptation to services modelled or developed without any adaptation possibility in mind and independently of specific usage contexts.

Context management and adaptation logic can be embedded into design-time models and can therefore be managed and reused more flexibly. At runtime *Apto* is able to generate an adaptive service corresponding to the new context. However, *Apto* does not address the instance migration issue which has been already addressed in the literature.

In its current implementation *Apto* lacks a validation tool for consistency check (in case one or more evolution fragments have to be applied to the original service) to ensure that this will not lead to a dead-lock and thus produce a valid service. Additionally the approach takes advantage of the MDD to transfer the service model and automatically generate the adapted service; however, it assumes that the service has been already modelled which is not always the case.

On the other hand, the service model represents BPEL at the syntactic level. While being separated, the evolution fragments and evolution primitives are low-level and service specific. In order to ease the design of service variants that need to be sound on a business-level, the *Apto* idea could be extended to be applied to the business-level model as well. In this case, the service adaptation takes part in two areas: (i) generating the adapted abstract business-level service, and (ii) transforming this service model into different concrete service artefacts according to different infrastructures or requirements.

Moreover, although the case study provides evidence of the usefulness of the approach from the design/implementation perspective, other evaluations in terms of the results gathered during the usage of the approach and the tool by a number of users and the number of cases the approach is able to cover are also needed.

Finally, *Apto* must be extended to accommodate more complex service variants scenarios. That is, if there are many dependencies between different evolution fragments that compose a variant, this has to be reflected in a semantic way. For example, if EF1 requires EF2, EF2 requires EF5, …, EF5 requires EFn, to efficiently resolve this EFs dependency *Apto* should be extended to model the dependency between evolution fragments using a semantic language.

9.2 Conclusions and Main Contributions

9.2.1 Conclusions

Despite the success and impressive research progress in the pervasive computing field, there are still problems and challenges to address which continue to be a major factor hindering the wide-spread adoption of a pervasive computing paradigm and therefore applicability. From a technical perspective, the reason is largely due to the difficulty the developer experiences in developing context-aware applications and adapting these applications to meet the specific needs of the user. The research during this study was based on the observation that existing approaches and tools are weak in providing a mechanism to adapt services at an adequately deep level and with sufficient automation. In addition, the existing approaches for context management are weak in providing a generic and robust context management infrastructure that facilitates the task of acquiring the context information related to the user and available in different domains they visit.

The study aims towards a software engineering approach which takes into consideration the ease of developing context-aware services. The following work has been undertaken during the study:

Approach

Based on the successful application of existing technologies, such as MDD, Jabber protocol, generative programming, and software product line, a new approach has been proposed to facilitate context modelling and management in distributed pervasive computing environment, situation recognition, as well as to adapt the context-aware services.

Implemented Prototypes

As a proof of concepts of the proposed approach, different prototypes have been developed to support the distributed domain-based context management and service adaptation, and to demonstrate and evaluate their applicability. In the design phase, after creating the service model, linkage model, evolution model, and context model, the *Apto* is able to automatically generate and deploy at runtime the new service definition corresponding to the current context. This is achieved by using the *Apto* implemented algorithm and the model-to-code transformation techniques imported from MDE.

The context model can be populated by using the *ubique* context management middleware services. This can be done either by configuring the context feature model or by specifying the context query which could span multiple domains. To implement the *ubique* approach, Jabber protocol has been leveraged to build upon and use its communication benefits. Individual context managers are deployed as context servers and their responsibility is limited to a specific domain. A collection of drivers for sensors and sensor agents for multiple purposes have been implemented which have been used to test a number of context aware applications. The aim is to make accessible the user’s context information related to the domains they visit.

Case Studies

Four case studies (Chapter 8) have been undertaken to illustrate and evaluate the usability, correctness, and applicability of the proposed approach, in terms of its capability of building context-aware adaptive service applications. These applications are able to define context queries that span one or more domains and to specify the context features they are interested in.

9.2.2 Contributions

The proposed approach in this study enables application developers to design and implement context-aware adaptive services. From a development point of view the original contribution of this thesis is the automation and deep level adaptation of services. From the context acquisition point of view the original contribution is a middleware infrastructure that enables application developers to specify the context information they need even if it is distributed in different domains. The key technique contributions are summarised below:

Product line Based Context Model

This study presents an approach for context-aware service development based on a flexible product line based context model which significantly enhances reusability of context information by providing context variability constructs (i.e. context features) to satisfy different application needs. This approach allows the context modeller to represent context in a high-level and in a more intuitive way and to improve overall systems performance.

Domain-based Context Management Middleware

The architecture of *ubique* hides the increasing complexity of context management from application developers and incorporates advanced mechanisms for the support of mobile users. In *ubique*, the classification and storage of the context information is performed in distributed context tuple spaces hosted in different context servers the hierarchy of which reflects the geographical structure of the physical world. A key point in *ubique* is that for each piece of monitored context information (contextlet) only one copy is maintained at a central point of access, the home domain. Additionally *ubique* meets most of the requirements presented in Section 3.2.1.

Process Mining Based Contextual Situation Recognition

The situation recognition approach focuses on the potential use of process mining techniques for measuring situation alignment, i.e., comparing the real situations of users with the expected situations. This approach is both based on and takes advantage of the *ubique* in order to reason about user's behaviour (situation) which may span different domains. It has been shown that approximate matching between an expected behaviour and the observed one requires a form of similarity measurement for comparing them. To this end, different process mining techniques have been leveraged to mine the user behaviour and to match it with the expected one. The approach has been shown to be effective at identifying contextual situations within pervasive computing applications.

MDD-based Mechanism for Context-aware Adaptive Services

The *Apto* approach proposed here aims to apply an adaptation to services modelled without any adaptation possibility in mind and independently of specific usage contexts. The notion of an evolution fragment and evolution primitive to capture the variability has been introduced. Finally, the *Apto* approach intends to support the viewpoint of context-aware adaptation as a crosscutting concern with respect to the core “business logic” of the service. In this way, the design of the service core can be decoupled from the design of the adaptation logic.

9.3 Future Work

Further research plans involve exploring the usage of the proposed approach in more complex scenarios; thus several points should be considered:

1- Currently, the idea of the product line based context model is applied for the context information available in the local domain the context server manages. In order to extend the proposed approach to the distributed context management architecture, two main points have to be addressed. Firstly, for the purpose of interoperability, we need a formal common semantics for context feature models managed by different context managers.

Secondly, the user has to be involved in determining the context information the application is allowed to acquire from different domains. Therefore, the user should be able to specify this information in her privacy policy. To this end, the system should allow the user to have several configurations of the context feature model available in each domain. In each configuration, which corresponds to a specific privacy policy, the user determines which context features are allowed to be acquired by the application and during which time period(s). This way, any application access to the user context information available in any domain would be controlled according to the privacy policy corresponding to that domain. This can be achieved either by asking the user to configure the context feature model or to use already-saved configurations. Then the context manager middleware is able to eliminate all context features that are not allowed to be acquired by the application.

Furthermore, as aforementioned, since the communication resources are limited, instant dissemination of the distributed context to the HDS cannot be achieved when the volume of data or the rate of change is high. One possible solution is to make a trade-off; that is to fine tune the context dissemination precision by specifying the time window interval to acquire the context or the threshold of the change value. The objective in the future work is to find mechanisms to include this tuning in the context feature model. This way, the application will be able to choose the degree of performance desired by configuring the context feature model.

2- In *ubique*, a Jabber-based context information dissemination protocol has been adopted. The storage and dissemination of the context information is performed by dissemination between distributed CSs. However, further research has to be done which involves exploring the use of the *ubique* middleware in more complex scenarios, extending *ubique* to support the geographic location based access to context information, the extension of the privacy protection scheme to consider not only specified domains but also domain types (e.g. a restaurant or a swimming pool), and a *ubique* extension to support context queries on the basis of the entities' and domains' types.

3- While the study of process mining based situation recognition revealed that process mining techniques are effective tools for recognizing situations, there are even more complex monitoring scenarios that need to be considered. In particular, the proposed approach needs to be extended to perform accurate situation recognition and tracking for environments that house multiple residents.

In addition, the proposed approach deals with mining the control flow, which is only one perspective addressed in process mining. Such information as the timestamp of a state or its subject (the person having experienced this state) can be used to derive high-level information about the process also in other perspectives. For example, the resource perspective looks at the set of users involved in the process, and their relationships. The social perspective can generate the social network, which may highlight different relationships between the users involved in the process. Therefore, the future work aims at leveraging these perspectives for providing users with more personalized services in pervasive environments, and integrating this work with *Apto* tool to provide a comprehensive software engineering framework for situation-aware systems.

4- The *Apto* tool could be enhanced by a graphical user interface which facilitates creating the evolution models and link the evolution fragments to the service elements. In addition, the *Apto* idea will be implemented as an extension to WebSphere Process Server V7 to take advantage of the instance migration feature.

On the other hand, in order to achieve the possibility of making deep changes to the service definition, in future work the *Apto* approach will be extended to regroup different service views' models. In this case, automated tools are needed to verify the integrity of the changes in the different views and generate the adapted service variant artefacts accordingly. Finally, the dependency between evolution fragments will be modelled using a semantic language to realise their dependency resolution.

References

[1] M. Román, C. Hess, R. Cerqueira, and R. H. Campbell, “A Middleware Infrastructure for Active Spaces,” *IEEE Pervasive Computing*, vol. 1(4), pp. 74-83, 2002.

[2] A. Chan and S.-N. Chuang, “MobiPADS: a reflective middleware for context-aware mobile computing,” *IEEE Transactions on Software Engineering*, vol. 29, no. 12, pp. 1072-1085, Dec. 2003.

[3] M. Grossmann, M. Bauer, N. Hönle, U.-P. Käppeler, D. Nicklas, and T. Schwarz, “Efficiently Managing Context Information for Large-Scale Scenarios,” in *Third IEEE International Conference on Pervasive Computing and Communications*, 2005, no. PerCom, pp. 331-340.

[4] J. I. Hong and J. A. Landay, “architecture for privacy-sensitive ubiquitous computing,” in *2nd International Conference on Mobile Systems, Applications, and Services*, 2004, vol. p, pp. 177-189.

[5] S. L. Kiani, M. Riaz, S. Lee, and Y.-K. Lee, “Context Awareness in Large Scale Ubiquitous Environments with a Service Oriented Distributed Middleware Approach,” in *Fourth Annual ACIS International Conference on Computer and Information Science (ICIS’05)*, 2005, vol. 5, pp. 513-518.

[6] A. Dearle et al., “Architectural Support for Global Smart Spaces,” in *Lecture Notes In Computer Science; Vol. 2574. Proceedings of the 4th International Conference on Mobile Data Management*, 2003, pp. 153-164.

[7] G. Chen, M. Li, and D. Kotz, “Data-centric middleware for context-aware pervasive computing,” *Pervasive and Mobile Computing*, vol. 4, no. 2, pp. 216-253, 2008.

[8] S. W. Loke, “Representing and reasoning with situations for context-aware pervasive computing: a logic programming perspective,” *The Knowledge Engineering Review*, vol. 19, no. 3, pp. 213-233, Jun. 2005.

[9] C. Bettini et al., “A survey of context modelling and reasoning techniques,” *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161-180, Jun. 2009.

[10] S. W. Loke, “On representing situations for context-aware pervasive computing : six ways to tell if you are in a meeting,” in *Proceedings of PerCom Workshops*, 2006, pp. 1-5.

[11] A. Ranganathan and R. H. Campbell, “An infrastructure for context-awareness based on first order logic,” *Personal and Ubiquitous Computing*, vol. 7, no. 6, pp. 353-364, Dec. 2003.

[12] S. S. Yau and J. Liu, “Hierarchical Situation Modeling and Reasoning for Pervasive Computing,” in *In Proceedings of the the Fourth IEEE Workshop on Software Technologies For Future Embedded and Ubiquitous Systems, and the Second international Workshop on Collaborative Computing, integration, and Assurance (Seus-Wccia’06)*, 2006, pp. 5-10.

[13] T. van Kasteren, A. Noulas, G. Englebienne, and B. Kröse, “Accurate activity recognition in a home setting,” *Proceedings of the 10th international conference on Ubiquitous computing - UbiComp ’08*, p. 1, 2008.

[14] J. Modayil, T. Bai, and H. Kautz, “Improving the recognition of interleaved activities,” in *Proceedings of the 10th international conference on Ubiquitous computing - UbiComp ’08*, 2008, p. 40.

[15] S. Mckeever, J. Ye, L. Coyle, C. Bleakley, and S. Dobson, “Activity recognition using temporal evidence theory,” *Journal of Ambient Intelligence and Smart Environments*, vol. 2, no. 3, pp. 253-269, 2010.

[16] O. Brdiczka, J. L. Crowley, and P. Reignier, “Learning Situation Models for Providing Context-Aware Services,” *Proceedings of Universal Access in Human-Computer Interaction, UAHCI 2007, in: Lecture Notes in Computer Science*, vol. 4555, pp. 23-32, 2007.

[17] P. Palmes, H. K. Pung, T. Gu, W. Xue, and S. Chen, “Object relevance weight pattern mining for activity recognition and segmentation☆,” *Pervasive and Mobile Computing*, vol. 6, no. 1, pp. 43-57, Feb. 2010.

[18] G. M. Kapitsaki, G. N. Prezerakos, N. D. Tselikas, and I. S. Venieris, “Context-aware service engineering: A survey,” *Journal of Systems and Software*, vol. 82, no. 8, pp. 1285-1297, Aug. 2009.

[19] T. Asikainen, T. Mannisto, and T. Soininen, “Kumbang: A domain ontology for modelling variability in software product families,” *Advanced Engineering Informatics*, vol. 21, no. 1, pp. 23-40, 2007.

[20] M. Koning, C. Sun, M. Sinnema, and P. Avgeriou, “VxBPEL: Supporting variability for Web services in BPEL☆,” *Information and Software Technology*, vol. 51, no. 2, pp. 258-269, Feb. 2009.

[21] M. Weiser, “The Computer for the 21st Century,” *Communications*, vol. 3, no. 3, pp. 3-11, 1991.

[22] E. Niemelä and J. Latvakoski, “Survey of requirements and solutions for ubiquitous software,” in *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia - MUM ’04*, 2004, no. October, pp. 71-78.

[23] A. K. Dey, “Understanding and Using Context,” *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4-7, 2001.

[24] K. Boukadi, C. Ghedira, S. Chaari, L. Vincent, and E. Bataineh, “CWSC4EC: how to employ context, web service, and community in enterprise collaboration,” in *Proceedings of the 8th international conference on new technologies in distributed systems*, 2008, pp. 27-38.

[25] T. Winograd, “Architectures for Context,” *Human-Computer Interaction*, vol. 16, no. 2, pp. 401-419, 2001.

[26] V. Viera, P. Brézillon, A. C. Salgado, and P. Tedesco, “A Context-Oriented Model for Domain-Independent Context Management,” *Revue d’intelligence artificielle*, vol. 22, no. 5, pp. 609-627, Oct. 2008.

[27] W. X. Htb, “The Context Gateway : A Pervasive Computing Infrastructure for Context Aware Services,” *Report submitted to School of Computing, National University of Singapore & Context -Aware Dept., Institute for Infocomm Research*, 2003.

[28] R. C. A. D. Rocha, M. Endler, and T. S. de Siqueira, “Middleware for ubiquitous context-awareness,” in *Proceedings of the 6th international workshop on Middleware for pervasive and ad-hoc computing - MPAC ’08*, 2008, pp. 43-48.

[29] R. C. A. da Rocha, “Context Management for Distributed and Dynamic Context-Aware Computing,” *PhD Thesis*, 2009.

[30] R. J. Orr and G. D. Abowd, “The Smart Floor : A Mechanism for Natural User Identification and Tracking,” in *Conference on Human Factors in Computing Systems*, 2000, pp. 275 - 276.

[31] H. Chen, T. Finin, and A. Joshi, “An Ontology for Context-Aware Pervasive Computing Environments,” *The Knowledge Engineering Review*, vol. 18, no. 3, pp. 197-207, 2004.

[32] A. K. Dey, G. D. Abowd, and D. Salber, “A conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications,” *Human-Computer Interaction*, vol. 16, no. 2, pp. 97-166, 2001.

[33] X. H. Wang, T. Gu, D. Q. Zhang, and H. K. Pung, “Ontology Based Context Modeling and Reasoning using OWL,” in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, 2004, vol. pp, pp. 18-22.

[34] N. Weienberg, A. Voisard, and R. Gartmann, “Using ontologies in personalized mobile applications,” in *Proceedings of the 12th annual ACM international workshop on Geographic information systems - GIS ’04*, 2004, no. 1, p. 2.

[35] F. Wang and K. J. Turner, “Towards personalised home care systems,” in *Proceedings of the 1st ACM international conference on PErvasive Technologies Related to Assistive Environments - PETRA ’08*, 2008.

[36] T. Gu, H. K. Pung, and D. Q. Zhang, “A middleware for building context-aware mobile services,” in *IEEE 59th Vehicular Technology Conference.*, 2004, pp. 2656-2660.

[37] B. Schilit, N. Adams, and R. Want, “Context-aware computing applications,” in *Workshop on Mobile Computing Systems and Applications*, 1995, pp. 85-90.

[38] G. Klyne et al., *Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0*. 2004, p. W3C Recommendation.

[39] J. Indulska, R. Robinson, A. Rakotonirainy, and K. Henricksen, “Experiences in Using CC/PP in Context-Aware Systems,” in *Proceedings of the 4th international Conference on Mobile Data Management*, 2003, pp. 247-261.

[40] A. Held, D. Ag, S. Buchholz, and A. Schill, “Modeling of Context Information for Pervasive Computing Applications,” in *Proceedings of SCI 2002/ISAS 2002*, 2002, pp. 1-6.

[41] Http://www.openmobilealliance.org/Technical/wapindex.aspx, “WAPFORUM. User Agent Profile (UAProf).” .

[42] D. Ayed, D. Delanote, and Y. Berbers, “MDD Approach for the Development of Context-Aware Applications,” *Lecture Notes in Computer Science*, vol. 4635/2007, pp. 15-28, 2007.

[43] K. Henricksen, J. Indulska, and A. Rakotonirainy, “Generating context management infrastructure from highlevel context models,” in *In Industrial Track proceedings of the 4th International Conference on Mobile Data Management (MDM2003)*, 2003, pp. 1-6.

[44] J. Indulska and R. Robinson, “Modelling Weiser’s ‘Sal’ scenario with CML,” in *The 6th Workshop on Context Modelling and Reasoning (CoMoRea’09) affiliated with IEEE PerCom'09*, 2009, pp. 1-6.

[45] T. Halpin, *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*. 2001.

[46] A. Schmidt and K. V. Laerhoven, “How to Build Smart Appliances,” *Ieee Personal Communications*, no. August, pp. 66-71, 2001.

[47] K. Cheverst, “Design of an object model for a context sensitive tourist GUIDE,” *Computers & Graphics*, vol. 23, no. 6, pp. 883-891, Dec. 1999.

[48] V. Akman and M. Surav, “The Use of Situation Theory in Context Modeling,” *Computational Intelligence*, vol. 13, no. 3, pp. 427-438, Aug. 1997.

[49] P. Gray and D. Salber, “Modelling and Using Sensed Context Information in the Design of Interactive Applications,” *In LNCS 2254: Proceedings of 8th IFIP International Conference on Engineering for Human-Computer Interaction (EHCI 2001)*, no. 1, pp. 317-335, 2001.

[50] G. Castelli, A. Rosi, M. Mamei, and F. Zambonelli, “A Simple Model and Infrastructure for Context-Aware Browsing of the World,” in *Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom’07)*, 2007, pp. 229-238.

[51] D. Nicklas, M. Großmann, T. Schwarz, S. Volz, and B. Mitschang, “A Model-Based , Open Architecture for Mobile , Spatially Aware Applications,” *Lecture Notes in Computer Science*, vol. 2121, pp. 117-135, 2001.

[52] N. Cipriani, M. Wieland, M. Großmann, and D. Nicklas, “Tool support for the design and management of context models,” *Information Systems*, Jul. 2010.

[53] J. Coutaz, J. L. Crowley, S. Dobson, and D. Garlan, “Context is Key,” *Communications of the ACM*, vol. 48, no. 3, pp. 49-53, 2005.

[54] “Web Ontology Language (OWL) homepage,” *http://www.w3.org/2004/OWL/*. .

[55] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, “SWRL: A Semantic Web Rule Language Combining OWL and RuleML,” *W3C Member Submission 21 May 2004*, no. 3, 2004.

[56] W. M. P. V. D. Aalst, H. T. D. Beer, and B. F. V. Dongen, “Process Mining and Verification of Properties : An Approach based on Temporal Logic,” in *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005*, 2005, pp. 130-147.

[57] T. Strang and C. Linnhoff-Popien, “A Context Modeling Survey,” in *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing*, 2004.

[58] S. W. LOKE, “Representing and reasoning with situations for context-aware pervasive computing : a logic programming perspective,” *The Knowledge Engineering Review*, vol. 19, no. 3, pp. 213-233, 2004.

[59] T. Springer, P. Wustmann, I. Braun, W. Dargie, and M. Berger, “A Comprehensive Approach for Situation-Awareness based on Sensing and Reasoning about Context,” in *Proceedings of the 5th international conference on Ubiquitous Intelligence and Computing*, 2008, pp. 143 - 157.

[60] I. McCowan, D. Gatica-Perez, S. Bengio, G. Lathoud, M. Barnard, and D. Zhang, “Automatic analysis of multimodal group actions in meetings.,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 3, pp. 305-17, Mar. 2005.

[61] W. Qin, Y. Shi, and Y. Suo, “Ontology-Based Context-Aware Middleware for Smart Spaces,” *Tsinghua Science & Technology*, vol. 12, no. 6, pp. 707-713, Dec. 2007.

[62] P. Reignier, O. Brdiczka, D. Vaufreydaz, J. L. Crowley, and J. Maisonnasse, “Context-aware environments: from specification to implementation,” *Expert Systems*, vol. 24, no. 5, pp. 305-320, 2007.

[63] J. Sun, Y. Zhang, and K. He, “A Petri-Net Based Context Representation in Smart Car Environment,” *Lecture Notes in Computer Science*, vol. 6104, no. 0302-9743, pp. 162-173, 2010.

[64] J. F. Allen, “Maintaining knowledge about temporal intervals,” *Communications of the ACM*, vol. 26, no. 11, pp. 832-843, Nov. 1983.

[65] H. Schmidt and F. J. Hauck, “SAMProc : Middleware for Self-adaptive Mobile Processes in Heterogeneous Ubiquitous Environments Categories and Subject Descriptors,” in *MDS ’07: Proceedings of the 4th on Middleware doctoral symposium*, 2007.

[66] V. Issarny, M. Caporuscio, and N. Georgantas, “A Perspective on the Future of Middleware-based Software Engineering A Perspective on the Future of Middleware-based Software Engineering,” in *FOSE ’07: 2007 Future of Software Engineering*, 2007, pp. 244 - 258.

[67] P. T. Eugster, P. a Felber, R. Guerraoui, and A.-M. Kermarrec, “The many faces of publish/subscribe,” *ACM Computing Surveys*, vol. 35, no. 2, pp. 114-131, Jun. 2003.

[68] P. Wyckoff, S. Mclaughry, T. Lehman, and D. Ford, “T Spaces,” *IBM Systems Journal*, vol. 37, no. 3, 1998.

[69] N. K. Mukhi, R. Konuru, and F. Curbera, “Cooperative middleware specialization for service oriented architectures,” in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters - WWW Alt. ’04*, 2004, p. 206.

[70] V. Grassi and A. Sindico, “Towards model driven design of service-based context-aware applications,” *International workshop on Engineering of software services for pervasive environments in conjunction with the 6th ESEC/FSE joint meeting - ESSPE ’07*, no. 11, pp. 69-74, 2007.

[71] Z. Maamar, D. Benslimane, and N. C. Narendra, “What can context do for web services?,” *Communications of the ACM*, vol. 49, no. 12, pp. 98-103, Dec. 2006.

[72] P. Bellavista and A. Corradi, *The Handbook of Mobile Middleware*. Auerbach Publications, 2006.

[73] L. O. B. Silva Santos, M. van Sinderen, and L. F. Pires, “Dynamic service discovery and composition for ubiquitous networks applications,” in *Proceedings of the 2006 ACM CoNEXT conference on - CoNEXT ’06*, 2006.

[74] D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha, “Service Composition for Mobile Environments,” *Mobile Networks and Applications*, vol. 10, no. 4, pp. 435-451, Aug. 2005.

[75] I. Jørstad, D. V. Thanh, and S. Dustdar, “Personalisation of Future Mobile Services,” in *9th International Conference on Intelligence in service delivery Networks*, 2004.

[76] S. M. Sadjadi, P. K. McKinley, and B. H. C. Cheng, “Transparent shaping of existing software to support pervasive and autonomic computing,” in *Proceedings of the 2005 workshop on Design and evolution of autonomic application software - DEAS ’05*, 2005.

[77] J. Kramer and J. Magee, “Self-Managed Systems : an Architectural Challenge,” in *Proceeding FOSE ’07 2007 Future of Software Engineering*, 2007.

[78] S. Smanchat, S. Ling, and M. Indrawan, “A survey on context-aware workflow adaptations,” in *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia - MoMM ’08*, 2008, p. 414.

[79] V. Andrikopoulos et al., *S-CUBE project: State of the art report on software engineering design knowledge and Survey of HCI and contextual Knowledge*, vol. 215483. 2008, pp. 1-115.

[80] G. Chafle, K. Dasgupta, A. Kumar, S. Mittal, and B. Srivastava, “Adaptation in Web Service Composition and Execution,” in *Proceedings of International Conference on Web Services (ICWS)*, 2006.

[81] J. Camara, C. Canal, J. Cubo, and J. Murillo, “An Aspect-Oriented Adaptation Framework for Dynamic Component Evolution1,” *Electronic Notes in Theoretical Computer Science*, vol. 189, pp. 21-34, Jul. 2007.

[82] R. Mietzner and F. Leymann, “Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors,” *2008 IEEE International Conference on Services Computing*, pp. 359-366, 2008.

[83] A. Carton, S. Clarke, A. Senart, and V. Cahill, “Aspect-Oriented Model-Driven Development for Mobile Context-Aware Computing,” in *First International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments (SEPCASE ’07)*, 2007.

[84] Q. Z. Sheng, S. Pohlenz, J. Yu, H. S. Wong, A. H. H. Ngu, and Z. Maamar, “ContextServ: A platform for rapid and flexible development of context-aware Web services,” *2009 IEEE 31st International Conference on Software Engineering*, no. Mdd, pp. 619-622, May. 2009.

[85] “Model-Driven Architecture homepage,” *http://www.omg.org/mda*. .

[86] W. Vanderaalst et al., “Business process mining: An industrial application,” *Information Systems*, vol. 32, no. 5, pp. 713-732, Jul. 2007.

[87] W. M. P. V. D. Aalst and B. F. V. Dongen, “ProM 4 . 0 : Comprehensive Support for Real,” pp. 484-494, 2007.

[88] W. M. P. V. D. Aalst et al., “ProM 4 . 0 : Comprehensive Support for Real Process Analysis,” vol. 4546, no. 2007, pp. 484-494, 2007.

[89] A. Rozinat, S. Zickler, M. Veloso, and W. M. P. V. D. Aalst, “Analyzing Multi-agent Activity Logs Using Process Mining Techniques,” in *Distributed Autonomous Robotic System 8*, 2009, pp. 251-260.

[90] V. Rubin, C. W. G, W. M. P. V. D. Aalst, E. Kindler, B. F. van Dongen, and W. Schafer, “Process Mining Framework for Software Processes,” in *Proceedings of the 2007 international conference on Software process*, 2007, pp. 19-20.

[91] B. F. V. Dongen, A. K. A. D. Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. V. D. Aalst, “The ProM Framework : A New Era in Process Mining Tool Support,” vol. 3536, no. 2005, pp. 444-454, 2005.

[92] L. M. Northrop, “SEI’s software product line tenets,” *IEEE Software*, vol. 19, no. 4, pp. 32-40, Jul. 2002.

[93] A. van der Hoek, “Design-time product line architectures for any-time variability,” *Science of Computer Programming*, vol. 53, no. 3, pp. 285-304, Dec. 2004.

[94] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and S. A. Peterson, “Feature-oriented domain analysis (FODA) - feasibility study,” *Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University*, 1990.

[95] P. Saint-Andre, “Extensible Messaging and Presence Protocol (XMPP): Core,” *http://www.ietf.org/rfc/rfc3920.txt*, 2004. .

[96] “XMPP Standards Foundation (XSF). XMPP Extensions,” *http://xmpp.org/xmpp-protocols/xmpp-extensions/*, 2010. .

[97] M. Roman, C. Hess, R. Cerqueira, a Ranganathan, R. H. Campbell, and K. Nahrstedt, “A middleware infrastructure for active spaces,” *IEEE Pervasive Computing*, vol. 1, no. 4, pp. 74-83, Oct. 2002.

[98] T. Strang, C. Linnhoff-popien, and K. Frank, “CoOL: A Context Ontology Language to enable Contextual Interoperability,” *4 th IFIP Int. Conf. on Distributed Applications and Interoperable Systems, (DAIS*, vol. vol, pp. 2893pp236-247, 2003.

[99] E. Christopoulou and A. Kameas, “GAS Ontology: An ontology for collaboration among ubiquitous computing devices,” *International Journal of Human-Computer Studies*, vol. 62, no. 5, pp. 664-685, May. 2005.

[100] D. Preuveneers et al., “Towards an extensible context ontology for Ambient Intelligence,” in *2nd European Symposium on Ambient Intelligence (EUSAI 2004)*, 2004, pp. 148–159.

[101] P. Moore, B. Hu, X. Zhu, W. Campbell, and M. Ratcliffe, “A Survey of Context Modeling for Pervasive Cooperative Learning,” in *IEEE Proceedings of the First International Symposium on Information Technologies and Applications in Education (ISITAE ’07)*, 2007, pp. K5-1-K5-6.

[102] J. Ye, L. Coyle, S. Dobson, and P. Nixon, “Ontology-based models in pervasive computing systems,” *The Knowledge Engineering Review*, vol. 22, no. 4, pp. 315-347, 2007.

[103] R. Krummenacher, H. Lausen, T. Strang, and J. Kopecky, “Analyzing the Modeling of Context with Ontologies,” in *International Workshop on Context-Awareness for Self-Managing Systems (Devices, Applications and Networks) - CASEMANS 2007 - as part of Pervasive 2007*, 2007, pp. 11 - 22.

[104] Z. Jaroucheh, X. Liu, and S. Smith, “CANDEL: Product Line Based Dynamic Context Management for Pervasive Applications,” in *International Conference on Complex, Intelligent and Software Intensive Systems (ARES/CISIS 2010)*, 2010, pp. 209-216.

[105] H. Chen, F. Perich, T. Finin, and A. Joshi, “Soupa: Standard Ontology for Ubiquitous and Pervasive Applications,” in *International Conference on Mobile and Ubiquitous Systems: Networking and Services*, 2004.

[106] H. Chen, T. Finin, and A. Joshi, “Semantic Web in the context broker architecture,” in *Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. Proceedings of the*, 2004, pp. 277-286.

[107] D. Connolly, F. V. Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, “DAML+OIL (March 2001) Reference Description,” *W3C Note 18 December 2001*, p. http://www.w3.org/TR/daml+oil-reference, 2001.

[108] T. Strang, C. Linnhoff-Popien, and K. Frank, “CoOL: Context Ontology Language to enable Contextual Interoperability,” *Lecture Notes in Computer Science*, vol. 2893, pp. 236-247, 2003.

[109] M. Kifer, G. Lausen, and J. Wu, “Logical Foundations of Object-Oriented and Languages,” *Journal of ACM*, vol. 42, no. 4, pp. 741–843, 1995.

[110] X. Wang, J. S. Dong, C. Chin, and S. R. Hettiarachchi, “Semantic Space: an infrastructure for smart spaces,” *IEEE Pervasive Computing*, vol. 3, no. 3, 2004.

[111] M. A. Strimpakou, I. G. Roussaki, and M. E. Anagnostou, “A Context Ontology for Pervasive Service Provision,” in *20th International Conference on Advanced Information Networking and Applications - Volume 1 (AINA’06)*, 2006, pp. 775-779.

[112] T. Gu, X. H. Wang, H. K. Pung, and D. Q. Zhang, “An Ontology-based Context Model in Intelligent Environments,” in *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, 2004.

[113] M. Khedr, “ACAI: agent-based context-aware infrastructure for spontaneous applications,” *Journal of Network and Computer Applications*, vol. 28, no. 1, pp. 19-44, Jan. 2005.

[114] “European IST-FP6 project MUSIC (Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environments),” *http://ist-music.eu*. .

[115] R. Reichle et al., “A Comprehensive Context Modeling Framework for Pervasive Computing Systems,” *DISTRIBUTED APPLICATIONS AND INTEROPERABLE SYSTEMS, Lecture Notes in Computer Science*, vol. 5053, pp. 281-295, 2008.

[116] R. Reichle et al., “A Context Query Language for Pervasive Computing Environments,” *2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 434-440, Mar. 2008.

[117] T. Kindberg and A. Fox, “System Software for Ubiquitous Computing,” *Pervasive Computing, IEEE*, vol. 1, pp. 70–81, 2002.

[118] M. Valla et al., “The Context API in the OMA Next Generation Service Interface,” in *Proceedings of ICIN 2010*, 2010.

[119] Z. Jaroucheh, X. Liu, and S. Smith, “Recognize contextual situation in pervasive environments using process mining techniques,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 2, no. 1, pp. 53-69, Dec. 2010.

[120] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam, “Middleware for Distributed Context-Aware Systems,” in *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA. Proceedings of the OTM Confederated International Conferences: CoopIS, DOA and ODBASE 2005, Part 1.*, 2005, vol. 3760, pp. 846-863.

[121] D. Lee and R. Meier, “A hybrid approach to context modelling in large-scale pervasive computing environments,” *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE - COMSWARE ’09*, p. 1, 2009.

[122] M. Strohbach, M. Bauer, E. Kovacs, C. Villalonga, and N. Richter, “Context Sessions – A Novel Approach for Scalable Context Management in NGN Networks,” in *MNCNA ’07 Proceedings of the 2007 Workshop on Middleware for next-generation converged networks and applications*, 2007, pp. 1-6.

[123] P. Floreen et al., “Towards a Context Management Framework for MobiLife,” in *In IST Mobile & Wireless Communications Summit*, 2005.

[124] M. Klemettinen, *Enabling Technologies for Mobile Services: The MobiLife Book*. 2007.

[125] J. Zebedee, P. Martin, K. Wilson, and W. Powley, “An Adaptable Context Management Framework for Pervasive Computing,” in *Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability*, 2009, pp. 114-146.

[126] G. Percivall, C. Reed, and J. Davidson, *Open Geospatial Consortium Inc . OGC White Paper OGC ® Sensor Web Enablement : Overview And High Level Architecture .*, no. December. 2007, pp. 1-14.

[127] G. Castelli and F. Zambonelli, “Contextual Data Management and Retrieval : a Self-organized Approach,” in *2009 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2009, pp. 535-538.

[128] D. Zhang, M. Guo, J. Zhou, D. Kang, and J. Cao, “Context reasoning using extended evidence theory in pervasive computing environments,” *Future Generation Computer Systems*, vol. 26, no. 2, pp. 207-216, Feb. 2010.

[129] X. Hong, C. Nugent, M. Mulvenna, S. McClean, B. Scotney, and S. Devlin, “Evidential fusion of sensor data for activity recognition in smart homes,” *Pervasive and Mobile Computing*, vol. 5, no. 3, pp. 236-252, Jun. 2009.

[130] J. Ye, L. Coyle, S. Dobson, and P. Nixon, “Using situation lattices in sensor analysis,” in *2009 IEEE International Conference on Pervasive Computing and Communications*, 2009, pp. 1-11.

[131] A. Erradi and P. Maheshwari, “AdaptiveBPEL : a Policy-Driven Middleware for Flexible Web Services Composition,” in *Proceedings of Middleware for Web Services (MWS) 2005*, 2005, pp. 5-12.

[132] A. Charfi and M. Mezini, “AO4BPEL: An Aspect-oriented Extension to BPEL,” *World Wide Web*, vol. 10, no. 3, pp. 309-344, 2007.

[133] F. Casati, S. Ilnicki, L. Jin, K. Vasudev, and M.-C. Shan, “Adaptive and Dynamic Service Composition in eFlow,” in *CAISE 2000*, 2000, pp. 13-31.

[134] O. Ezenwoye and S. M. Sadjadi, “TRAP/BPEL: A framework for dynamic adaptation of composite services,” in *In Proceedings of the International Conference on Web Information Systems and Technologies (WEBIST 2007*, 2007.

[135] a Erradi and P. Maheshwari, “A broker-based approach for improving Web services reliability,” in *IEEE International Conference on Web Services (ICWS’05)*, 2005, vol. 1, pp. 355-362.

[136] M. Reichert, S. Rechtenbach, A. Hallerbach, and T. Bauer, “Extending a Business Process Modeling Tool with Process Configuration Facilities: The Provop Demonstrator,” in *BPM’09 Demonstration Track, Business Process Management Conference*, 2009, vol. 1.

[137] J. Choi, Y. Cho, K. Shin, and J. Choi, “A Context-Aware Workflow System for Dynamic Service Adaptation,” in *Computational Science and Its Applications – ICCSA 2007*, 2007, pp. 335-345.

[138] R. M, U. Greiner, and E. Rahm, “A G E N T W O R K : a workflow system supporting rule-based workflow adaptation,” *Data & Knowledge Engineering*, vol. 51, no. 0169-023X, pp. 223-256, 2004.

[139] P. Costanza, R. Hirschfeld, and O. Nierstrasz, “Context-oriented Programming,” *Journal of Object Technology*, vol. 7, no. 3, pp. 125-151, 2008.

[140] K. Czarnecki, C. Hwan, and K. T. Kalleberg, “Feature Models are Views on Ontologies,” in *Proceedings of the 10th International on Software Product Line Conference*, 2006, vol. 1, pp. 41-51.

[141] J. Man, A. Yang, and X. Sun, “Shared Ontology for Pervasive Computing,” *Lecture Notes in Computer Science*, vol. 3818, pp. 64 - 78, 2005.

[142] M. Baldauf, S. Dustdar, and F. Rosenberg, “A survey on context-aware systems,” *Int. J. Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263-277, 2007.

[143] A. Mehrotra, *GSM System Engineering*. Mobile Communications Series, Artech House Publishers., 1997.

[144] I. Roussaki, M. Strimpakou, C. Pils, N. Kalatzis, and N. Liampotis, “Distributed Context Management in Support of Multiple Remote Users,” in *Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability*, 2009, pp. 84-113.

[145] XMPP, “XMPP Standards Foundation,” *http://www.xmpp.org/*, 2004. .

[146] OpenFire, “OpenFire Server,” *http://www.igniterealtime.org/projects/openfire/index.jsp*, 2010. .

[147] Z. Jaroucheh, X. Liu, and S. Smith, “An approach to domain-based scalable context management architecture in pervasive environments,” *Personal and Ubiquitous Computing*, vol. 1617-4917, no. 3, pp. 1-15, Jun. 2011.

[148] S. Dobson and J. Ye, “Using fibrations for situation identification,” in *Proceedings of Pervasive 2006 Workshops, Springer*, 2006.

[149] A. Dahlbom, L. Niklasson, G. Falkman, and A. Loutfi, “Towards template-based situation recognition,” *Proceedings of SPIE Symposium Defense, Security, and Sensing*, vol. 7352, 2009.

[150] W. van Der Aalst, T. Weijters, and L. Maruster, “Workflow mining: discovering process models from event logs,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128-1142, Sep. 2004.

[151] B. F. V. Dongen and W. M. P. V. D. Aalst, “Multi-phase Process mining: Building Instance Graphs,” in *International Conference on Conceptual Modeling (ER 2004), volume 3288 of Lecture Notes in Computer Science.*, 2004, pp. 362-376.

[152] A. J. M. M. Weijters and W. M. P. V. D. Aalst, “Rediscovering Workflow Models from Event-Based Data using Little Thumb,” *Integrated Computer-Aided Engineering*, vol. 10, no. 2 (April 2003), pp. 151 - 162, 2003.

[153] A. Rozinat and W. M. P. V. D. Aalst, “Conformance Testing : Measuring the Fit and Appropriateness of Event Logs and Process Models,” in *BPM 2005 Workshops (Workshop on Business Process Intelligence*, 2006, pp. 163-176.

[154] T. van Kasteren, A. Noulas, G. Englebienne, and B. Kröse, “Accurate activity recognition in a home setting,” in *Proceedings of the 10th international conference on Ubiquitous computing - UbiComp ’08*, 2008.

[155] A. Jehad Sarkar, Y.-K. Lee, and S. Lee, “A Smoothed Naive Bayes-Based Classifier for Activity Recognition,” *IETE Technical Review*, vol. 27, no. 2, pp. 107-119, 2010.

[156] J. Yu, Q. Z. Sheng, P. Falcarin, and M. Morisio, “Weaving Business Processes and Rules: A Petri Net Approach,” in *Information Systems: Modeling, Development, and Integration, Third International United Information Systems Conference, UNISCON 2009*, 2009, pp. 121-126.

[157] H. Tran, U. Zdun, and S. Dustdar, “View-based and Model-driven Approach for Reducing the Development Complexity in Process-Driven SOA,” in *Internaltional Working Conference on Business Process and Services Computing (BPSC’07)*, 2007, pp. 105-124.

[158] A. Hallerbach, T. Bauer, and M. Reichert, “Managing process variants in the process life cycle,” in *10th Int’l Conf. on Enterprise Information Systems (ICEIS'08)*, 2008.

[159] Microsoft-MSDN, “XAML Overview,” *http://msdn.microsoft.com/en-us/library/ms752059.aspx*, 2010. .

[160] D. Chappell, “Introducing Microsoft Windows Workflow Foundation: An Early Look,” *http://msdn.microsoft.com/en-us/library/aa480215.aspx*, 2005. .

[161] Z. Jaroucheh, X. Liu, and S. Smith, “Apto : A MDD-based Generic Framework for Context-aware Deeply Adaptive Service-based Processes,” in *ICWS, The IEEE 8th International Conference on Web Services*, 2010, pp. 219 - 226.

Appendix A : Abbreviations and Acronyms

All the abbreviations and acronyms used in this thesis are defined below.

Abbreviation/Acronyms	**Description**
AOP	Aspect Oriented Programming.
BPEL	Business Process Execution Language.
BPEL4WS	Business Process Execution Language for Web Services.
HTTP	Hyper Text Transfer Protocol.
OMG	Object Management Group.
OOP	Object Oriented Programming.
QoS	Quality of Service.
SOAP	Simple Object Access Protocol.
SPL	Software Product Line.
UDDI	Universal Description, Discovery and Integration.
UML	Unified Modelling Language.
WSDL	Web Service Description Language.
RDF	Resource Description Framework.
OWL	Web Ontology Language.
CC/PP	Composite Capabilities/Preference Profile.
MDA	Model Driven Architecture.
MDD	Model Driven Development.
OCM	Ontology-based Context Model

Appendix B : Ontology-based Context Model

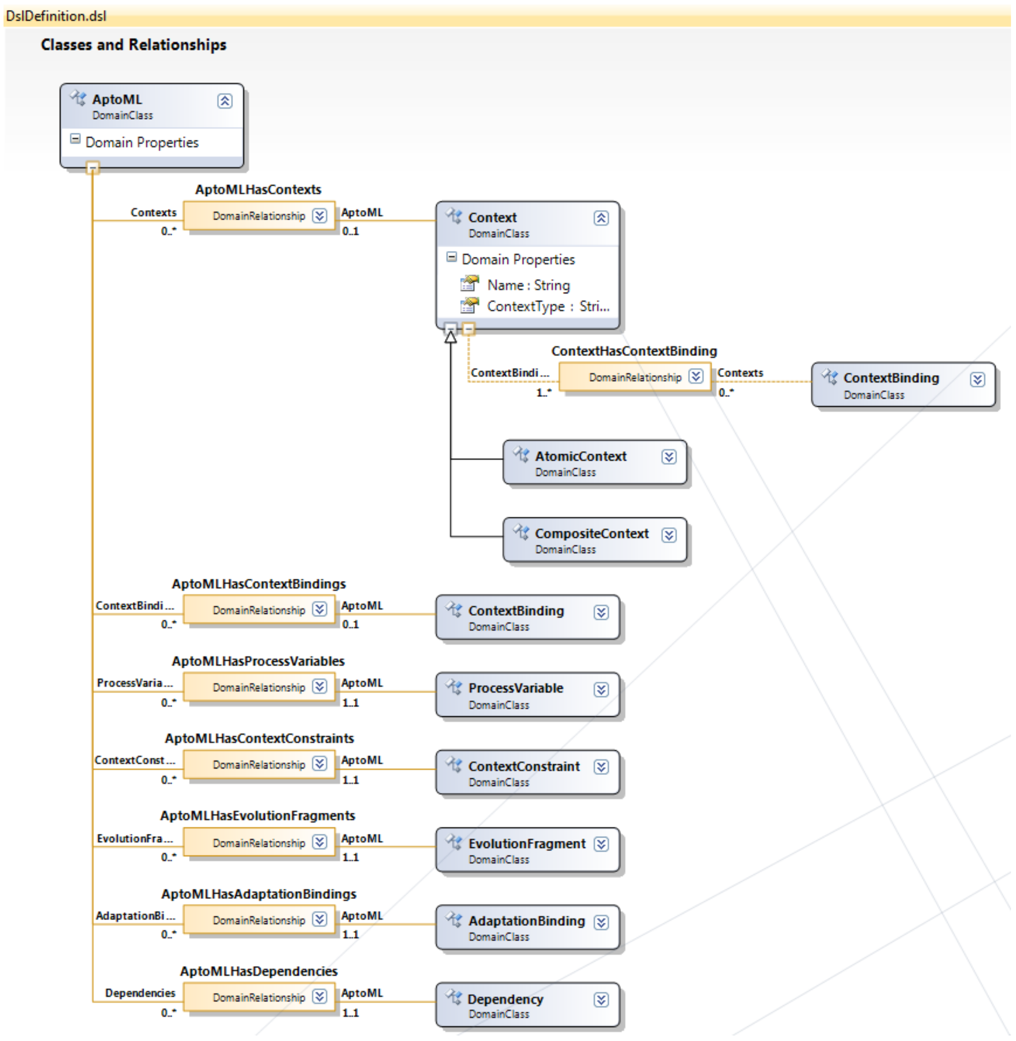
```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:owl="http://www.w3.org/2002/07/owl#"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns="http://www.napier.ac.uk/candel#"
 xml:base="http://www.napier.ac.uk/candel">



<!-- Classes -->

  <owl:Class rdf:ID="FMConf">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:label>Always</rdfs:label>
  </owl:Class>

  <!-- Person Related Classes -->

  <owl:Class rdf:ID="ContactInformation">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:label>ContactDetails</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="PhDStudent">
    <rdfs:subClassOf rdf:resource="Researcher"/>
    <rdfs:label>PhDStudent</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="ExpertResearcher">
    <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <rdfs:label>Experts</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="Researcher">
    <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <rdfs:label>Publications</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="PersonFillsPresenterRole">
    <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <rdfs:label>CurrentRole</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="PersonFillsSessionChairRole">
    <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <rdfs:label>CurrentRole</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="ParticipantOfPresentationHappeningNow">
    <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <rdfs:label>CurrentRole</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="PresenterOfPresentationHappeningNow">
    <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <rdfs:label>CurrentRole</rdfs:label>
  </owl:Class>


  <!-- Place Related Classes -->

  <owl:Class rdf:ID="Place">
    <rdfs:label>Location</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="AtomicPlace">
    <rdfs:subClassOf rdf:resource="#Place"/>
    <rdfs:label>RoomResolution</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="CompoundPlace">
    <rdfs:subClassOf rdf:resource="#Place"/>
    <rdfs:label>BuildingResolution</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="Building">
    <rdfs:subClassOf rdf:resource="#CompoundPlace"/>
    <rdfs:label>BuildingResolution</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="Campus">
    <rdfs:subClassOf rdf:resource="#CompoundPlace"/>
    <rdfs:label>BuildingResolution</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="Room">
    <rdfs:subClassOf rdf:resource="#AtomicPlace"/>
    <rdfs:label>RoomResolution</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="OtherPlaceInBuilding">
    <rdfs:subClassOf rdf:resource="#AtomicPlace"/>
    <rdfs:label>Location</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="MeetingRoom">
    <rdfs:subClassOf rdf:resource="#Room"/>
    <rdfs:label>RoomResolution</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="RoomHasPresentationHappeningNow">
    <rdfs:subClassOf rdf:resource="#Room"/>
    <rdfs:label>RoomResolution</rdfs:label>
    <rdfs:label>CurrentRole</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Journal">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:label>Publications</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="APlusJournal">
  <rdfs:subClassOf rdf:resource="#Journal"/>
  <rdfs:label>Publications</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="BPlusJournal">
  <rdfs:subClassOf rdf:resource="#Journal"/>
  <rdfs:label>Publications</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="CPlusJournal">
  <rdfs:subClassOf rdf:resource="#Journal"/>
  <rdfs:label>Publications</rdfs:label>
</owl:Class>


<!-- Event Related Classes -->

<owl:Class rdf:ID="Artefact">
  <rdfs:subClassOf rdf:resource="http://xmlns.com/wordnet/1.6/Document"/>
  <rdfs:label>Publications</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Paper">
  <rdfs:subClassOf rdf:resource="#Artefact"/>
  <rdfs:label>Publications</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Award">
  <rdfs:subClassOf rdf:resource="#Artefact"/>
  <rdfs:label>ExpertHavingAwards</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Event">
  <rdfs:subClassOf rdf:resource="http://xmlns.com/wordnet/1.6/Event-1"/>
  <rdfs:label>Conference</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="OrganisedEvent">
  <rdfs:subClassOf rdf:resource="#Event"/>
  <rdfs:label>Conference</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="AcademicEvent">
  <rdfs:subClassOf rdf:resource="#OrganisedEvent"/>
  <rdfs:label>Conference</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="ConferenceEvent">
  <rdfs:subClassOf rdf:resource="#AcademicEvent"/>
  <rdfs:label>Conference</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="SessionEvent">
  <rdfs:subClassOf rdf:resource="#AcademicEvent"/>
  <rdfs:label>Conference</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="PaperSession">
  <rdfs:subClassOf rdf:resource="#SessionEvent"/>
  <rdfs:label>Conference</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="PosterSession">
   <rdfs:subClassOf rdf:resource="#SessionEvent"/>
   <rdfs:label>Conference</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="TalkEvent">
  <rdfs:subClassOf rdf:resource="#AcademicEvent"/>
  <rdfs:label>Conference</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="PaperPresentation">
  <rdfs:subClassOf rdf:resource="#TalkEvent"/>
  <rdfs:label>CurrentRole</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="PaperPresentationHappeningNow">
  <rdfs:subClassOf rdf:resource="#PaperPresentation"/>
  <rdfs:label>CurrentRole</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="PosterPresentation">
  <rdfs:subClassOf rdf:resource="#TalkEvent"/>
  <rdfs:label>CurrentRole</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="PosterPresentationHappeningNow">
  <rdfs:subClassOf rdf:resource="#PaperPresentation"/>
  <rdfs:label>CurrentRole</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="NonAcademicEvent">
  <rdfs:subClassOf rdf:resource="#OrganisedEvent"/>
  <rdfs:label>Conference</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="BreakEvent">
```

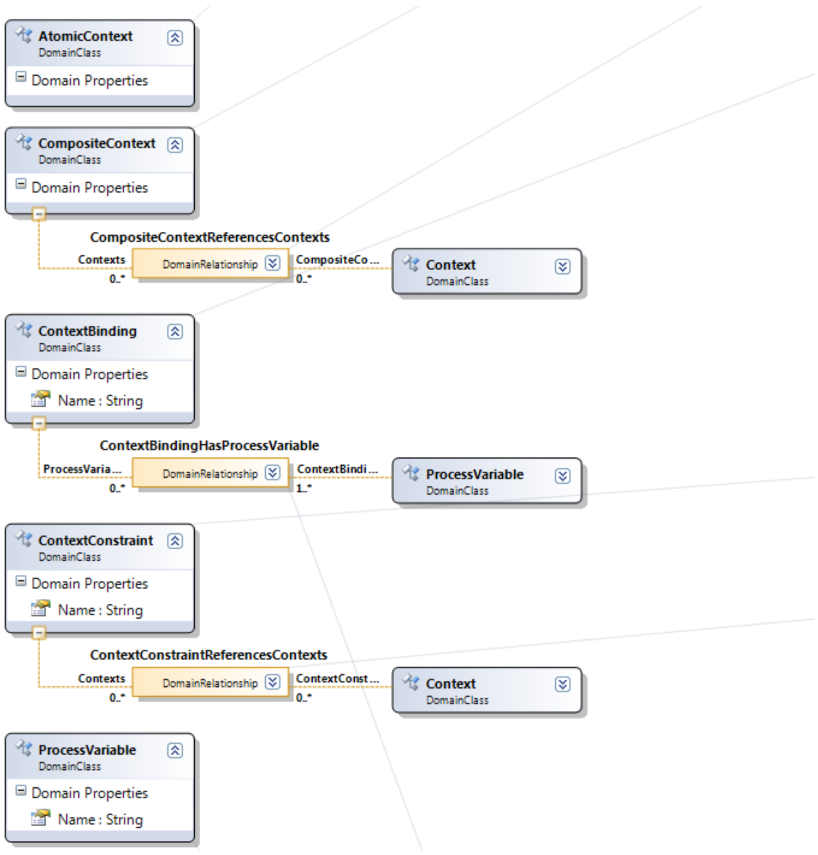
```xml
    <rdfs:subClassOf rdf:resource="#NonAcademicEvent"/>
    <rdfs:label>Conference</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="CoffeeBreak">
    <rdfs:subClassOf rdf:resource="#BreakEvent"/>
    <rdfs:label>Conference</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="TalkEventSchedule">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:label>Conference</rdfs:label>
  </owl:Class>



  <!-- Role Related Classes -->

  <owl:Class rdf:ID="Role">
   <rdfs:subClassOf rdf:resource="http://xmlns.com/wordnet/1.6/Role-1"/>
    <rdfs:label>CurrentRole</rdfs:label>
    <rdfs:label>StaticRole</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="ConferenceChair">
    <rdfs:subClassOf rdf:resource="#Role"/>
    <rdfs:label>StaticRole</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="OrganisingCommitteeMember">
    <rdfs:subClassOf rdf:resource="#Role"/>
    <rdfs:label>StaticRole</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="ProgrammeCommitteeMember">
    <rdfs:subClassOf rdf:resource="#Role"/>
    <rdfs:label>StaticRole</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="SessionChair">
    <rdfs:subClassOf rdf:resource="#Role"/>
    <rdfs:label>StaticRole</rdfs:label>
    <rdfs:label>StaticRole</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="Presenter">
    <rdfs:subClassOf rdf:resource="#Role"/>
    <rdfs:label>CurrentRole</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="Reviewer">
    <rdfs:subClassOf rdf:resource="#Role"/>
    <rdfs:label>StaticRole</rdfs:label>
  </owl:Class>


<!-- Object Properties -->

  <!-- Event Related Object Properties -->


  <owl:ObjectProperty rdf:ID="hasAttendee">
    <rdfs:domain rdf:resource="#OrganisedEvent"/>
    <rdfs:range rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <rdfs:label>ParticipatingPeople</rdfs:label>
    <!-- inverse of #attendeeAt -->
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="invitedSpeaker">
    <rdfs:domain rdf:resource="#TalkEvent"/>
    <rdfs:range rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <rdfs:label>ParticipatingPeople</rdfs:label>
    <!-- inverse of #attendeeAt -->
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="hasSchedule">
    <rdfs:domain rdf:resource="#PaperPresentation"/>
    <rdfs:range rdf:resource="#TalkEventSchedule"/>
    <rdfs:label>Conference</rdfs:label>
    <rdfs:label>CurrentRole</rdfs:label>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="relatedToEvent">
    <rdfs:domain rdf:resource="#Artefact"/>
    <rdfs:range rdf:resource="#AcademicEvent"/>
    <rdfs:label>Conference</rdfs:label>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="relatedToJournal">
    <rdfs:domain rdf:resource="#Artefact"/>
    <rdfs:range rdf:resource="#Journal"/>
    <rdfs:label>Publications</rdfs:label>
  </owl:ObjectProperty>


  <owl:ObjectProperty rdf:ID="hasLocation">
    <rdfs:domain rdf:resource="#OrganisedEvent"/>
    <rdfs:range rdf:resource="#Place"/>
    <rdfs:label>Location</rdfs:label>
    <!-- has inverse isLocationFor -->
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="hasRole">
    <rdfs:domain rdf:resource="#AcademicEvent"/>
    <rdfs:range rdf:resource="#Role"/>
    <rdfs:label>CurrentRole</rdfs:label>
    <rdfs:label>StaticRole</rdfs:label>
    <!-- inverse of #isRoleAt -->
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="hasArtefact">
    <rdfs:domain rdf:resource="#AcademicEvent"/>
```

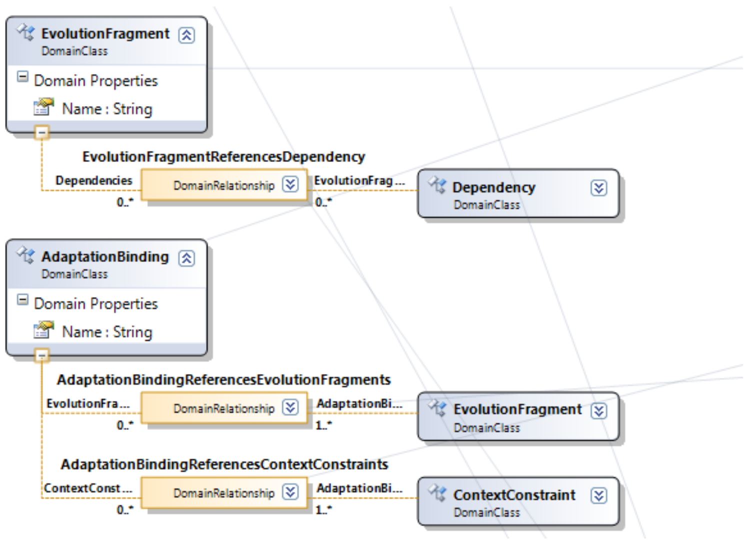
```xml
  <rdfs:range rdf:resource="#Artefact"/>
  <rdfs:label>BookChapter</rdfs:label>
  <rdfs:label>Paper</rdfs:label>
  <!-- inverse of #relatedToEvent -->
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasRelatedArtefact">
  <rdfs:domain rdf:resource="#Journal"/>
  <rdfs:range rdf:resource="#Artefact"/>
  <rdfs:label>ExpertHavingJournalPublications</rdfs:label>
  <!-- inverse of #relatedToEvent -->
</owl:ObjectProperty>


<!-- Place Related Object Properties -->

<owl:ObjectProperty rdf:ID="spatiallySubsumes">
  <rdfs:domain rdf:resource="#CompoundPlace"/>
  <rdfs:range rdf:resource="#Place"/>
  <rdfs:label>Location</rdfs:label>
  <rdfs:label>BuildingResolution</rdfs:label>
  <!-- inverse of #isSpatiallySubsumedBy -->
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="isSpatiallySubsumedBy">
  <rdfs:domain rdf:resource="#AtomicPlace"/>
  <rdfs:range rdf:resource="#CompoundPlace"/>
  <rdfs:label>Location</rdfs:label>
  <rdfs:label>BuildingResolution</rdfs:label>
  <!-- inverse of #spatiallySubsumes -->
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="isLocationFor">
  <rdfs:domain rdf:resource="#Place"/>
  <rdfs:range rdf:resource="#OrganisedEvent"/>
  <rdfs:label>Location</rdfs:label>
</owl:ObjectProperty>

<!-- Role Related Object Properties -->

<owl:ObjectProperty rdf:ID="isRoleAt">
  <rdfs:domain rdf:resource="#Role"/>
  <rdfs:range rdf:resource="#AcademicEvent"/>
  <rdfs:label>StaticRole</rdfs:label>
  <!-- has inverse #hasRole -->
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="heldBy">
  <rdfs:domain rdf:resource="#Role"/>
  <rdfs:range rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <rdfs:label>StaticRole</rdfs:label>
  <rdfs:label>CurrentRole</rdfs:label>
  <!-- has inverse #holdsRole -->
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="holdsRole">
  <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <rdfs:range rdf:resource="#Role"/>
  <rdfs:label>StaticRole</rdfs:label>
  <!-- has inverse #heldBy -->
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="isFilledBy">
  <rdfs:domain rdf:resource="#Role"/>
  <rdfs:range rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <rdfs:label>StaticRole</rdfs:label>
  <rdfs:label>CurrentRole</rdfs:label>
</owl:ObjectProperty>

<!-- Person Related Object Properties -->

<owl:ObjectProperty rdf:ID="participatesIn">
  <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <rdfs:range rdf:resource="#OrganisedEvent"/>
  <rdfs:label>ParticipatingPeople</rdfs:label>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="fillsRole">
  <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <rdfs:range rdf:resource="#Role"/>
  <rdfs:label>CurrentRole</rdfs:label>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasContactInformation">
  <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <rdfs:range rdf:resource="#ContactInformation"/>
  <rdfs:label>ContactDetails</rdfs:label>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="locatedInRoom">
  <rdfs:subPropertyOf rdf:resource="#locatedInAtomicPlace" />
  <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <rdfs:range rdf:resource="#Room"/>
  <rdfs:label>RoomResolution</rdfs:label>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="locatedInCompoundPlace">
  <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <rdfs:range rdf:resource="#CompoundPlace"/>
  <rdfs:label>BuildingResolution</rdfs:label>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="locatedInAtomicPlace">
  <rdfs:subPropertyOf rdf:resource="#locatedInCompoundPlace" />
  <rdfs:domain rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <rdfs:range rdf:resource="#AtomicPlace"/>
  <rdfs:label>BuildingResolution</rdfs:label>
</owl:ObjectProperty>


<owl:ObjectProperty rdf:ID="hasAward">
  <rdfs:domain rdf:resource="#Paper"/>
```

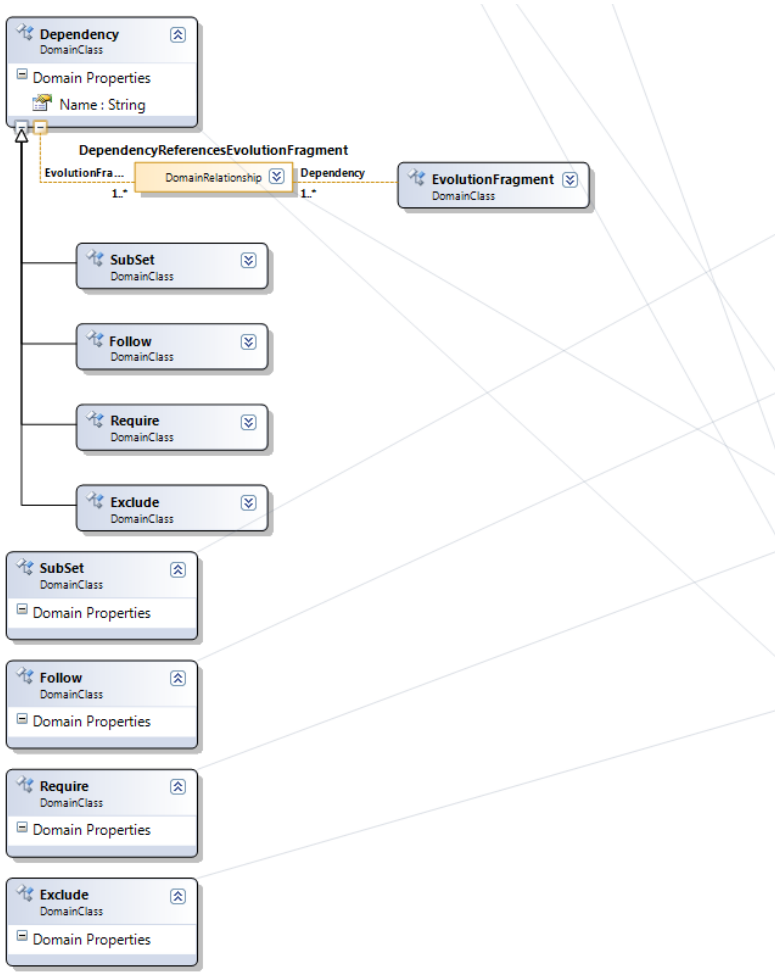
```
    <rdfs:range rdf:resource="#Award"/>
    <rdfs:label>ExpertHavingAwards</rdfs:label>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="authorOf">
    <rdfs:domain rdf:resource="#Researcher"/>
    <rdfs:range rdf:resource="#Paper"/>
    <rdfs:label>Publications</rdfs:label>
 </owl:ObjectProperty>


<!-- Datatype Properties -->

  <owl:DatatypeProperty rdf:ID="eventHasStartDateTime">
    <rdfs:domain rdf:resource="#OrganisedEvent"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:label>CurrentRole</rdfs:label>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="eventHasEndDateTime">
    <rdfs:domain rdf:resource="#OrganisedEvent"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:label>CurrentRole</rdfs:label>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="hasStartDateTime">
    <rdfs:domain rdf:resource="#PaperPresentation"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:label>CurrentRole</rdfs:label>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="hasEndDateTime">
    <rdfs:domain rdf:resource="#PaperPresentation"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:label>CurrentRole</rdfs:label>
  </owl:DatatypeProperty>


  <owl:DatatypeProperty rdf:ID="biblioReference">
    <rdfs:domain rdf:resource="#Artefact"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:label>Paper</rdfs:label>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="hasLatitude">
    <rdfs:domain rdf:resource="#Place"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdfs:label>Location</rdfs:label>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="hasLongitude">
    <rdfs:domain rdf:resource="#Place"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdfs:label>Location</rdfs:label>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="hasPrettyName">
    <rdfs:domain rdf:resource="#Place"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:label>Location</rdfs:label>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="hasName">
    <rdfs:stereotype>Publications</rdfs:stereotype>
    <rdfs:domain rdf:resource="#Journal"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:label>Publications</rdfs:label>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="hasRank">
    <rdfs:domain rdf:resource="#Journal"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdfs:label>ExpertHavingJournalPublications</rdfs:label>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="hasInfluenceIndex">
    <rdfs:domain rdf:resource="#Journal"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdfs:label>ExpertHavingJournalPublications</rdfs:label>
  </owl:DatatypeProperty>


  <owl:DatatypeProperty rdf:ID="minimumJournalRank">
    <rdfs:domain rdf:resource="#FMConf"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdfs:label>ExpertHavingJournalPublications</rdfs:label>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="minimumInfluenceIndex">
    <rdfs:domain rdf:resource="#FMConf"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdfs:label>ExpertHavingJournalPublications</rdfs:label>
  </owl:DatatypeProperty>

  <owl:ObjectProperty rdf:ID="topAwardName">
    <rdfs:domain rdf:resource="#FMConf"/>
    <rdfs:range rdf:resource="#Award"/>
    <rdfs:label>Publications</rdfs:label>
  </owl:ObjectProperty>




  <FMConf rdf:ID="FMConfiguration">
    <minimumJournalRank
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">100.0</minimumJournal
Rank>
    <minimumInfluenceIndex
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">10.0</minimumInfluenceI
ndex>
    <topAwardName rdf:resource="#IEEE_Award"/>
    <rdfs:label>Always</rdfs:label>
  </FMConf>

  <Award rdf:ID="IEEE_Award">
     <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </Award>

  <Award rdf:ID="ACM_Award">
     <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </Award>

  <ConferenceEvent rdf:ID="Conference1">
  </ConferenceEvent>

  <ConferenceEvent rdf:ID="Conference2">
  </ConferenceEvent>

  <ConferenceEvent rdf:ID="Conference3">
  </ConferenceEvent>


  <Journal rdf:ID="JournalOne">
   <hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">JOURNAL
OF THE ACM</hasName>
   <hasRank
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">204.0</hasRank>
   <hasInfluenceIndex
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">15.83</hasInfluenceInde
x>
  </Journal>

  <Journal rdf:ID="JournalTwo">
   <hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">ACM
COMPUTING SURVEYS</hasName>
   <hasRank
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">382.0</hasRank>
   <hasInfluenceIndex
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">3.45</hasInfluenceIndex
>
  </Journal>

  <Journal rdf:ID="JournalThree">
   <hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">ACM
TRANSACTIONS ON COMPUTER SYSTEMS</hasName>
   <hasRank
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">462.0</hasRank>
   <hasInfluenceIndex
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">2.78</hasInfluenceIndex
>
  </Journal>

  <Journal rdf:ID="JournalFour">
   <hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">IEEE-
ACM TRANSACTIONS ON NETWORKING</hasName>
   <hasRank
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">669.0</hasRank>
   <hasInfluenceIndex
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">16.40</hasInfluenceInde
x>
  </Journal>


<Paper rdf:ID="FirstPaper">
                        <hasAward rdf:resource="#IEEE_Award"/>
                        <hasAward rdf:resource="#ACM_Award"/>
                        <relatedToJournal rdf:resource="#JournalOne"/>
                        <biblioReference
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Product Line based
Context Modelling </biblioReference>
 </Paper>

  <Paper rdf:ID="SecondPaper">
                        <hasAward rdf:resource="#IEEE_Award"/>
                        <relatedToJournal rdf:resource="#JournalThree"/>
                        <biblioReference
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"> Second Paper Title
</biblioReference>
  </Paper>

  <Paper rdf:ID="ThirdPaper">
                        <relatedToEvent rdf:resource="#Conference2"/>
                        <biblioReference
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"> Third Paper
Title</biblioReference>
  </Paper>

  <Researcher rdf:ID="Alice">
    <locatedInRoom rdf:resource="#C33"/>
    <authorOf rdf:resource="#FirstPaper"/>
    <authorOf rdf:resource="#SecondPaper"/>
  </Researcher>

  <Researcher rdf:ID="Bob">
                        <authorOf rdf:resource="#FirstPaper"/>
  </Researcher>

  <Researcher rdf:ID="John">
                        <authorOf rdf:resource="#ThirdPaper"/>
  </Researcher>

  <Building rdf:ID="ComputingBuilding">
  </Building>

  <Building rdf:ID="CivilEngineeringBuilding">
  </Building>

  <Building rdf:ID="MedicineBuilding">
  </Building>

  <Room rdf:ID="CThirty">
                        <isSpatiallySubsumedBy rdf:resource="#ComputingBuilding"/>
  </Room>
  <Room rdf:ID="CThirtyOne">
                        <isSpatiallySubsumedBy rdf:resource="#ComputingBuilding"/>
  </Room>
  <Room rdf:ID="CThirtyTwo">
                        <isSpatiallySubsumedBy rdf:resource="#ComputingBuilding"/>
  </Room>
  <Room rdf:ID="CThirtyThree">
                        <isSpatiallySubsumedBy rdf:resource="#ComputingBuilding"/>
  </Room>

  <Room rdf:ID="EFifty">
                        <isSpatiallySubsumedBy
rdf:resource="#CivilEngineeringBuilding"/>
  </Room>
  <Room rdf:ID="EFiftyOne">
                        <isSpatiallySubsumedBy
rdf:resource="#CivilEngineeringBuilding"/>
  </Room>
  <Room rdf:ID="EFiftyTwo">
                        <isSpatiallySubsumedBy
rdf:resource="#CivilEngineeringBuilding"/>
  </Room>
  <Room rdf:ID="EFiftyThree">
                        <isSpatiallySubsumedBy
rdf:resource="#CivilEngineeringBuilding"/>
  </Room>

  <PaperPresentation rdf:ID="FirstPaperPresentation">
    <hasStartDateTime
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2010-03-
22GMT10:00:00</hasStartDateTime>
    <hasEndDateTime
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">2010-03-
22GMT21:00:00</hasEndDateTime>
    <!-- <invitedSpeaker rdf:resource="#Alice"/> -->
  </PaperPresentation>



<!-- OWL hacks -->
  <owl:Class rdf:about="http://swrc.ontoware.org/ontology#ResearchTopic"/>
  <!-- <owl:Class rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
  <owl:Class rdf:about="http://www.w3.org/2001/XMLSchema#integer"/>-->
  <owl:Class rdf:about="http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing">
       <rdfs:label>Always</rdfs:label>
  </owl:Class>

  <owl:Class rdf:about="http://xmlns.com/foaf/0.1/Organisation">
       <rdfs:label>Always</rdfs:label>
  </owl:Class>

  <owl:Class rdf:about="http://xmlns.com/foaf/0.1/Person">
     <rdfs:label>Always</rdfs:label>
  </owl:Class>

  <owl:Class rdf:about="http://xmlns.com/wordnet/1.6/Announcement"/>

  <owl:Class rdf:about="http://xmlns.com/wordnet/1.6/Document">
      <rdfs:label>Always</rdfs:label>
  </owl:Class>

  <owl:Class rdf:about="http://xmlns.com/wordnet/1.6/Event-1">
       <rdfs:label>Always</rdfs:label>
  </owl:Class>

  <owl:Class rdf:about="http://xmlns.com/wordnet/1.6/Menu"/>
  <owl:Class rdf:about="http://xmlns.com/wordnet/1.6/Role-1">
       <rdfs:label>Always</rdfs:label>
  </owl:Class>

  <owl:Class rdf:about="http://xmlns.com/wordnet/1.6/Sponsorship"/>

  <owl:DatatypeProperty rdf:about="http://purl.org/dc/elements/1.1/contributor"/>
  <owl:DatatypeProperty rdf:about="http://purl.org/dc/elements/1.1/creator"/>
  <owl:DatatypeProperty rdf:about="http://purl.org/dc/elements/1.1/date"/>
  <owl:DatatypeProperty rdf:about="http://purl.org/dc/elements/1.1/description"/>
  <owl:DatatypeProperty rdf:about="http://purl.org/dc/elements/1.1/title"/>
<!-- -->

<!-- OWL/RDFS compatibility hacks by Denny Vrandecic
     (so RDFS only tools can handle OWL ontologies)
     deploy where necessary
The following three axioms provide a mapping of the OWL terms to the RDFS
terms. So
if a tool is not able to read the OWL ontology as it is, uncomment these axioms
(or better, load an ontology with only these three axioms and merge them) and if the
tool
fulfills the RDFS specification it will magically be able to deal with the whole
ontology.
Mind you, you may not add this tool to the OWL ontology, or else you move to OWL
Full.
-->
<!--
    <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Class">
      <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"
/>
    </owl:Class>

    <rdfs:Property rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty">
      <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Property" />
    </rdfs:Property>

    <rdfs:Property rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty">
      <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Property" />
    </rdfs:Property>
-->


</rdf:RDF>
```

Appendix C : AptoML Language




AtomicContext
DomainClass
Domain Properties
CompositeContext
DomainClass
Domain Properties
CompositeContextReferencesContexts
Contexts
0..*
DomainRelationship
CompositeCo...
0..*
Context
DomainClass
ContextBinding
DomainClass
Domain Properties
Name : String
ContextBindingHasProcessVariable
ProcessVaria...
0..*
DomainRelationship
ContextBindi...
1..*
ProcessVariable
DomainClass
ContextConstraint
DomainClass
Domain Properties
Name : String
ContextConstraintReferencesContexts
Contexts
0..*
DomainRelationship
ContextConst...
0..*
Context
DomainClass
ProcessVariable
DomainClass
Domain Properties
Name : String


EvolutionFragment
DomainClass
Domain Properties
Name : String
EvolutionFragmentReferencesDependency
Dependencies
0..*
DomainRelationship
EvolutionFrag...
0..*
Dependency
DomainClass
AdaptationBinding
DomainClass
Domain Properties
Name : String
AdaptationBindingReferencesEvolutionFragments
EvolutionFra...
0..*
DomainRelationship
AdaptationBi...
1..*
EvolutionFragment
DomainClass
AdaptationBindingReferencesContextConstraints
ContextConst...
0..*
DomainRelationship
AdaptationBi...
1..*
ContextConstraint
DomainClass


Dependency
DomainClass
Domain Properties
Name : String
DependencyReferencesEvolutionFragment
EvolutionFra...
DomainRelationship
Dependency
1..*
1..*
EvolutionFragment
DomainClass
SubSet
DomainClass
Follow
DomainClass
Require
DomainClass
Exclude
DomainClass
SubSet
DomainClass
Domain Properties
Follow
DomainClass
Domain Properties
Require
DomainClass
Domain Properties
Exclude
DomainClass
Domain Properties