Edinburgh Napier UNIVERSITY

A SEMANTIC FRAMEWORK FOR UNIFIED CLOUD SERVICE SEARCH, RECOMMENDATION, RETRIEVAL AND MANAGEMENT

DAREN FANG

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS OF EDINBURGH NAPIER UNIVERSITY, FOR THE AWARD OF

DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

EDINBURGH NAPIER UNIVERSITY

SEPTEMBER 2015

Abstract

Cloud computing (CC) is a revolutionary paradigm of consuming Information and Communication Technology (ICT) services. However, while trying to find the optimal services, many users often feel confused due to the inadequacy of service information description. Although some efforts are made in the semantic modelling, retrieval and recommendation of cloud services, existing practices would only work effectively for certain restricted scenarios to deal for example with basic and non-interactive service specifications. In the meantime, various service management tasks are usually performed individually for diverse cloud resources for distinct service providers. This results into significant decreased effectiveness and efficiency for task implementation. Fundamentally, it is due to the lack of a generic service management interface which enables a unified service access and manipulation regardless of the providers or resource types.

To address the above issues, the thesis proposes a semantic-driven framework, which integrates two main novel specification approaches, known as agility-oriented and fuzziness-embedded cloud service semantic specifications, and cloud service access and manipulation request operation specifications. These consequently enable comprehensive service specification by capturing the in-depth cloud concept details and their interactions, even across multiple service categories and abstraction levels. Utilising the specifications as CC knowledge foundation, a unified service recommendation and management platform is implemented. Based on considerable experiment data collected on real-world cloud services, the approaches demonstrate distinguished effectiveness in service search, retrieval and recommendation tasks whilst the platform shows outstanding performance for a wide range of service access, management and interaction tasks. Furthermore, the framework includes two sets of innovative specification processing algorithms specifically designed to serve advanced CC tasks: while the fuzzy rating and ontology evolution algorithms establish a manner of collaborative cloud service specification, the service orchestration reasoning algorithms reveal a promising means of dynamic service compositions.

Acknowledgments

I would like to thank my director of study *Dr. Xiaodong Liu*, supervisor *Dr. Imed Romdhani*, and my panel chair *Dr. Michael Smyth* for all their help, support, expertise and understanding throughout the period of my 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 research.

I am most indebted to my beloved wife, *Qian Liu*, for her endless support and understanding during this period. Finally, great appreciation and thanks to my parents and my parents-in-law. Although they are in China, they always encourage me over the phone and via email, 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 Fang D, Liu X, Romdhani I, Pahl C, Jamshidi P, (2015) An agility-oriented and fuzziness-embedded semantic model for collaborative cloud service search, retrieval and recommendation. *Future Generation Computer Systems* (Accepted).

2 Fang D, Liu X, Romdhani I, Pahl, C, (2015) An Approach to Unified Cloud Service Access, Manipulation and Dynamic Orchestration via Semantic Cloud Service Operation Specification Framework. *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 4, no. 14, pp. 1-20.

3 Fang D, Liu X, Liu L, Yang H, (2014) OCSO: Off-the-cloud service optimisation for green efficient service resource utilisation. *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 3, no. 9, pp. 1-17.

Conference papers

4 Fang D, Liu X, Romdhani I, (2014) A Loosely-coupled Semantic Model for Diverse and Comprehensive Cloud Service Search and Retrieval, *CLOUD COMPUTING 2014, The Fifth International Conference on Cloud Computing*, pp.6-11.

5 Fang D, Liu X, Liu L and Yang H, (2013) TARGO: Transition and Reallocation Based Green Optimisation for Cloud VMs, *IEEE International Conference on Green Computing and Communications (GreenCom)*, pp.215-223.

6 Fang D, Liu X, Romdhani I and Zhao H, (2012) Towards OWL2 Natively Supported Fuzzy Cloud Ontology, *36th Annual Computer Software and Applications Conference Workshops (COMPSACW)*, pp. 328-333.

7 Fang D, Liu X, Liu L and Yang H, (2012) Evolution for the sustainability of Internetware, *Internetware '12 Proceedings of the Fourth Asia-Pacific Symposium on Internetware*, no. 17.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Problem Statement

Cloud computing (CC) revolutionises the world's ICT with on-demand provisioning, pay-per-use self-service, ubiquitous network access and location-independent resource pooling. Its reliable, scalable and customisable computational service and resource provision can adapt rapidly and effectively to nearly all kinds of needs for all major industrial sectors [23, 92]. The rapid development incurs numerous new cloud services and service updates continuously. Facing the increasingly complex service market, cloud service consumers (CSCs) often need to dig deeply while searching for optimal services. Meanwhile, many cloud service providers (CSPs) provide unique management portals for their own services and resources [41]. The service interfaces, functionalities and operation environments are mostly diverse. Accordingly, while trying to manage multiple cloud services and resources, CSCs usually have to use a variety of cloud portals for different CSPs. This significantly limits the effectiveness and efficiency for tasks deployment and implementation [40].

In recent years, Web Ontology Language (OWL) 98 has been widely adopted for web service semantic specifications [105, 144]. The formal entity specification and reference framework can enable the integration of a wide range of aspects, e.g. context information [80], user requirements [73], business processes [72]. Indeed, this greatly assists service design, development, invocation and composition tasks in pervasive environments [107].

Although considerable research efforts are made to drive and enhance the interoperability and composition of cloud applications, services and resources [7, 59, 150], significant research gaps are found among the existing service reference frameworks and models. Consequently, these impose urgent needs yet great challenges on the specification and retrieval of cloud services,

whereas an effective cloud service recommendation and management tool is in demand for a variety of CSCs.

1.2 Motivation

As a series of cloud (service) semantic models propagate [150, 73, 93, 122, 144, 161], they still suffer from limitations. *Firstly*, the majority of the existing models cannot maintain comprehensive service information across multiple abstraction levels (i.e. Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS)). These models fail to reveal the various agile interactions among cloud services and resources of such matrix structure (e.g. SaaS services can be deployed on PaaS platforms whilst PaaS services may rely on IaaS resources). *Secondly*, a limited number of the models can effectively present the diverse full and potential service functions and features; none of them clarifies the range of connections or cooperation among cloud services and companies who have (hidden) relationships (e.g. some cloud services can orchestrate with others whilst some CSPs have certain industry relationships). *Thirdly*, most cloud services are “agile”, i.e. adaptable at run time in their functions, interfaces, capacity, etc (see detailed discussion in Section 4.1.2). Yet, these agility aspects are often ignored or poorly disclosed in the existing models. Consequently, the lack of these critical aspects causes ineffectiveness while implementing service search, discovery, retrieval, and recommendation tasks.

Meanwhile, to deal with the cloud (service) interoperability issues, a number of cloud (service and resource) interoperability and portability approaches are proposed. These solutions include but not limited to: open cloud API (Application Programming Interface) development such as jclouds [4], libcloud [5], fog [42]; service specifications such as TOSCA [150], mOSAIC [7]; unified cloud management protocols/drivers such as OCCI [107]. Yet, despite their capabilities of handling certain specific service/resource categories, it is difficult to find any that allows adequate management for diverse CSPs’ services and resources via a common interface. This is mainly due to the lack of a unified

service specification model that interprets cloud service and resource entities and deals with the interoperability among CSPs [146].

1.3 Aims and Objectives of the Research

Driven by the above motivations, the vision of the research is to provide comprehensive assistances for a combination of cloud service search, recommendation, retrieval, and management tasks. To a wider extent, this would consequently enhance service discovery, evaluation, invocation, manipulation and orchestration for additional usages. Accordingly, the aim of the thesis is to convey the diverse features, attributes and operation behaviours of cloud services to a unified semantic specification framework. In this thesis, the framework is defined as a layered structure which involves a combination of semantic modelling approaches, cloud service ontologies, relevant specification processing algorithms and mechanisms. Within the framework, relevant components are interrelated and work together to serve certain parts of the aim. More specifically, these are delivered through the following four objectives.

1.3.1 To Develop an Approach for Effective Cloud Service Search, Recommendation and Retrieval

In order to assist cloud service search, recommendation and retrieval tasks, a number of cloud service ontology models have been proposed. However, these ontologies cannot sustain comprehensive and in-depth cloud service specifications. Fundamentally, these ontologies lack of focus on the “agility” aspects existed widely for many cloud concepts and service entities. In addition, CC involves many vague definitions and descriptions that conventional semantic specifications cannot effectively handle, due to the fundamental OWL description logic consistency and simplicity reasons [11, 42].

The first objective will focus on a novel agility-oriented and fuzziness-embedded cloud service ontology. The agility-oriented design would allow the ontology to effectively specify comprehensive and in-depth cloud service specifications. The

OWL2 fuzzy extensions would enable adequate and precise specifications regarding the uncertainties involved in cloud service descriptions.

1.3.2 To Develop an Approach for Generic Service Remote Management

Despite a wide range of approaches proposed to deal with cloud service remote management and interoperability issues, they are only capable of handling certain limited cloud resource categories and operation types. Indeed, this is mainly due to their restricted model/framework designs that fail to support cloud services in a wider scale [40].

The second objective will address an innovative unified cloud service operation specification approach. It would capture adequate details for service operation view, retrieval and execution tasks. Moreover, the approach should enable functions for enhanced service operation assistance tasks (e.g., operation execution verifications, schedulers), even across distinct provider clouds. Consequently, this ought to drive accurate and efficient cloud service data/resource remote management tasks.

1.3.3 To Integrate the Cloud Service Specifications and Prototype Implementation

Currently, it is difficult to find any cloud service tool that is capable of delivering the combination of service search, recommendation, retrieval and management functions. Targeting such research gap, the third objective is to integrate the above research components and implement a versatile cloud service assistance prototype tool. With the aim of providing distinguished assistances for CSCs, it is designed to combine the above functions into a unified interface, i.e. an integrated cloud service recommendation and management platform.

Dynamic cloud service management operation tasks are performed via service API calls. This is to be addressed by a cloud service API mapping mechanism within the prototype. A new mapper component is proposed for invoking

appropriate service API requests whenever remote management operations are initiated by users.

1.3.4 To Conduct Case Studies and Evaluation

For proof-of-concept, validation and evaluation, the final objective is to apply a series of real-life case studies and experiments to critically examine the proposed approaches and tool implementations. Considering the wide range of assistance functions provided for diverse cloud service types and categories, the case studies would involve multiple cloud services from distinct service types/providers, whereas the experiment results ought to provide comparisons with typical solutions from existing research/industry practices.

1.4 Contributions to Knowledge

The research overcomes the existing limitations by offering comprehensive cloud service search, recommendation and retrieval functions for diverse service categories/types. Further, it closes the research gaps by providing a unified service interface for cloud service remote management tasks over multiple clouds. Accordingly, they result into a series of contributions: *Firstly*, a number of new modelling approaches are proposed. They provide an innovative means of cloud service semantic modelling towards precise and comprehensive CC entity specifications. *Secondly*, based on the approaches, two cloud service ontologies are developed as resourceful knowledge sources for CC specifications. The proposed ontologies are capable of describing the diversity of service data and specifications for real-world cloud services, regardless of their service types/categories/providers. *Thirdly*, within the unified cloud service assistance platform, a series of algorithms (i.e. fuzziness rating and operation reasoning algorithms) and mechanisms (i.e. ontology evolution and API mapping) are developed. They enable an effective means of service specification and interoperability enhancement for many advanced requirements and tasks. More specifically, the contributions are described as follows:

- **OWL2 Natively Supported Fuzzy Extension Approach**

The thesis proposes a novel OWL2 fuzzy extension approach which can be easily applied to ordinary OWL2 ontologies for fuzziness representation. According to the relevant PLN and fuzzy set and relationships theory, three categories of fuzzy scenarios are demonstrated to specifically deal with a certain type of fuzziness. By adopting the approach, various cloud service vagueness can be adequately revealed. This consequently enhances cloud service modelling by achieving precise specifications.

- **Service Access and Manipulation Operation Specification (SAMOS) Approach**

In contrast with other existing service operation specification framework and models, SAMOS provides a light-weight yet effective solution for comprehensive service operation specifications. Resting on ontological modelling specifications, it comprises complete specifications for service operations regardless of the service/operation/provider types. By decoupling complicated service operations into two categories of granular service operations, which are seen as service information requests (SIRs) and service manipulation requests (SMRs), it can effectively specifies all typical operation details including the parameters, requirement, outcome, condition changes, etc.

- **Agility-oriented and Fuzziness-embedded Cloud Service Ontology (AoFeCSO)**

By researching over two hundreds of real-world cloud services and using the above modelling techniques, two large scale cloud service ontologies are developed. In particular, for service search, recommendation and retrieval tasks, AoFeCSO provides comprehensive specifications for cloud service descriptions, functions, features, characteristics, etc. aspect. It adopts a loosely-coupled and agility-oriented design which maximally utilises the full range of OWL2 (latest) axiom assertions. Moreover, it is deployed as a fuzziness-embedded ontology that stays active, where certain specifications are asserted with fuzzy weights

and hence able to illustrate the hidden/inexplicit/controversial nature in the form of truth degrees.

- **Cloud Service Access and Manipulation Ontology (CSAMO)**

For cloud service operation specification towards generic service remote management requirement, CSAMO is developed based on the proposed modelling framework. It comprehensively describes the relevant cloud entities, their attributes and relationships involved in service operations. By preserving the complexity which lies behind the diversity of operation tasks, CSAMO effectively interprets and instructs cloud service access and manipulation operations in a formal systematic way.

- **Cloud Service Specification and Interoperability Enhancement Algorithms and Mechanisms**

Based on the above two cloud service specification ontologies and approaches, a tool namely cloud service recommendation and management platform (CSRMP) is implemented. The platform demonstrates a practical use of the proposed AoFeCSO and CSAMO by establishing a unified interface for diverse cloud service usage assistance tasks including service search, recommendation, management (plus additional comparison, evaluation and orchestration).

The platform owns a range of innovative algorithms and mechanisms that would greatly enhance cloud service specifications and interoperability. Specifically, a cloud service API mapping mechanism proposed within the platform provides wide compatibility with real IaaS, PaaS and SaaS services from multiple provider clouds. This allows CSCs to effectively search, view, create and amend a wide range of cloud services/resources/data via a unified structured interface. Moreover, a fuzziness rating management algorithm and an ontology evolution mechanism enable automatic and dynamic ontology evolution without interrupting concurrent service retrieval actions. Additionally, a series of service operation reasoning algorithms are capable of presenting intelligent dynamic assistances based on the analysis of real-time service data and user

requirements. These consequently achieve distinguished assistances for advanced cloud service usage tasks.

1.5 Research Method

The thesis adopts a combination of research methods including literature review and tool-based case studies.

Initially, comprehensive review of philosophical literature is undertaken with regard to CC semantic models, ontological specifications, OWL fuzzy extension, service operation specifications. Through in-depth review and analysis of the latest literatures, several issues and limitations are found on existing cloud (service) specifications models and relevant modelling techniques. These lead to the design and development of the series of novel approaches proposed subsequently.

To justify and evaluate the proposed modelling approaches and cloud service specification approaches, a prototype tool is implemented and a series of case studies are conducted. Utilising a number of distinct real-world cloud services, extensive experiments are conducted to evaluate the functionality, effectiveness, efficiency of proposed approaches.

Papers have been published based on research outcome at each milestone. This enables valuable assessments of the work from other researchers in terms of contribution and justification within the field.

1.6 The Structure of the Thesis

The thesis is organised as follows:

Chapter 1 outlines introduction of the research with the problem statement, the aim and objectives of the research, the contributions to knowledge and the statement of methodology.

Chapter 2 broadly reviews the relevant literature, including the background of CC, sematic modelling, existing approaches and tools for cloud services, and fuzzy logic theories and applications.

Chapter 3 discusses the related work in details focusing on three main research areas: cloud service specification models and recommendation tools, ontology fuzzy extensions, and unified cloud service management.

Chapter 4 presents the design and implementation of AoFeCSO, which is to enhance cloud service search, recommendation and retrieval tasks.

Chapter 5 demonstrates SAMOS approach, which is to enable and assist cloud service remote management and potential orchestration tasks via a generic interface.

Chapter 6 interconnects the previous research objectives by providing the architecture design and implementation of the proposed prototype: the integrated cloud service recommendation, retrieval, management and orchestration platform.

In Chapter 7, using popular real-world cloud services from multiple providers, a series of case studies and experiments are conducted to illustrate and evaluate the functions and performances of the proposed approach and tool.

Finally, Chapter 8 summarises the thesis by presenting the conclusions and the future work.

Chapter 2 Literature Review

To further explain the research questions and to formulate solutions, this chapter broadly reviews the relevant subject areas. These involve an overview of CC (the delivery models, the deployment types, the parties and roles, service characteristics, etc.) Then, they lead to a series of additional literature, including the current practises of cloud service modelling, service operations specifications, OWL fuzzy extension, service recommendation systems, etc. The above aspects are seen as the grounding where the proposed approaches are established and developed.

2.1 Background of Cloud Computing

2.1.1 Cloud Service Delivery Models

Infrastructure-as-a-Service (IaaS) [137] is defined as the service model that provides fundamental computing resources such as virtualised processing power, storage, networking systems, etc. Typical examples are seen as Amazon EC2 [1], Rackspace Cloud Servers [118] and GoGrid Cloud Servers [52]. IaaS model eliminates substantial IT investment for users whist it achieves an effective use of computing hardware for the providers [48]. Generally speaking, these services provide many types of customisability, such as the options of virtual machine (VM) configurations, operating systems (OSs), network configurations, supplied software, etc. [101]. Nevertheless, the service providers tend to maintain maximum control of all underlying hardware and the software kernel [86].

Platform-as-a-Service (PaaS) [93] refers to the service model which provisions virtualised hosting and development environment for users to run, test and deploy services/applications. Typical examples are known as Google AppEngine [55], Salesforce Service Clouds [127] and IBM SmartCloud [114]. PaaS platforms usually offer customisable environment feature and attribute controls, e.g. programming language historical version supports, resource

scaling functionalities, monitor and alarm features, etc. [29]. However, beyond those, users are often restricted for any further configurations (e.g. virtualisation hardware, architecture, OS, network setting, etc.) [128]

Software-as-a-Service (SaaS) [70] is the service model for cloud-enabled applications that are designed to achieve specific software-alike functions, e.g. Rackspace Cloud Load Balancers [117], Google Docs [56] and Cisco WebEx [26]. SaaS users are typically more concerned about what and how a service achieves certain application functions, rather than the underlying service provision details (e.g. virtualisation platform, virtualisation software) [48]. Most likely, very limited information (no more than prices, service features, service level agreements (SLAs), etc.) are disclosed to public; sensitive contents such as the cloud hardware, system and platform information of these services are hidden and not customisable [78]. SaaS eliminates the effort of licensing, installing, maintaining, and updating, compared with traditional software solutions [15].

2.1.2 Cloud Service Deployment Types

- *Public Cloud*

Public clouds [137] are recognised as the clouds where service resources are provided and maintained by third-party CSP(s) over the Internet. Typically, public cloud CSCs have little concerns for the underlying service provision details and technologies; instead, they tend to care more regarding the competent factors such as the services' SLA, features, quality of service (QoS), etc. offered by CSPs [8]. Further, CSCs usually have no/limited control over the fundamental cloud infrastructure/hardware, whereas their service management behaviours and records are often monitored as they consume the services [62].

- *Private Cloud*

Private clouds [137] are built, deployed, and managed privately by certain users. This means that computing hardware and software are owned and configured

privately within one's own networks. In contrast with public clouds, the deployment type exposes superior advantages for the customisable service design and implementation, few legal concerns, privately managed account/security/maintenance controls, etc. Yet, this also means that the owner has to spend more time, resource, efforts, etc. while managing the cloud [31].

- *Hybrid Cloud*

Hybrid cloud [137] stands for the solution that makes use of both public and private cloud resources to fulfil the computing needs. The deployment model effectively mitigates their individual weaknesses and therefore, improves the overall computing/resource performance. This flexible manner is considered to be more sensible while dealing with complex cases and needs, as many characteristics of public cloud and private cloud are complementary [137].

- *Community Cloud*

Community cloud [137] is run and controlled by a number of organisations which are of the same or similar interest. Between these organisations, data and policy occurred in the community cloud are often shared easily and securely, rather than crossing the entire Internet.

2.1.3 Parties and Roles

According to IBM [10] and NIST [88], CC involves two minimum parties known as CSP and CSC. CSP is regarded as the party who provides cloud services and resources and is responsible for the availability and QoS. CSC is defined as the party that requests and uses cloud services which are provided by CSPs. It can be a single person or an organisation. CSC can also be involved in the management of the service. For instance, IaaS users often manage the updates and settings of their virtual compute resources on their own.

IBM cloud reference architecture comprises a party in addition to the above parties, called is cloud service creator. The main role of it is to develop and

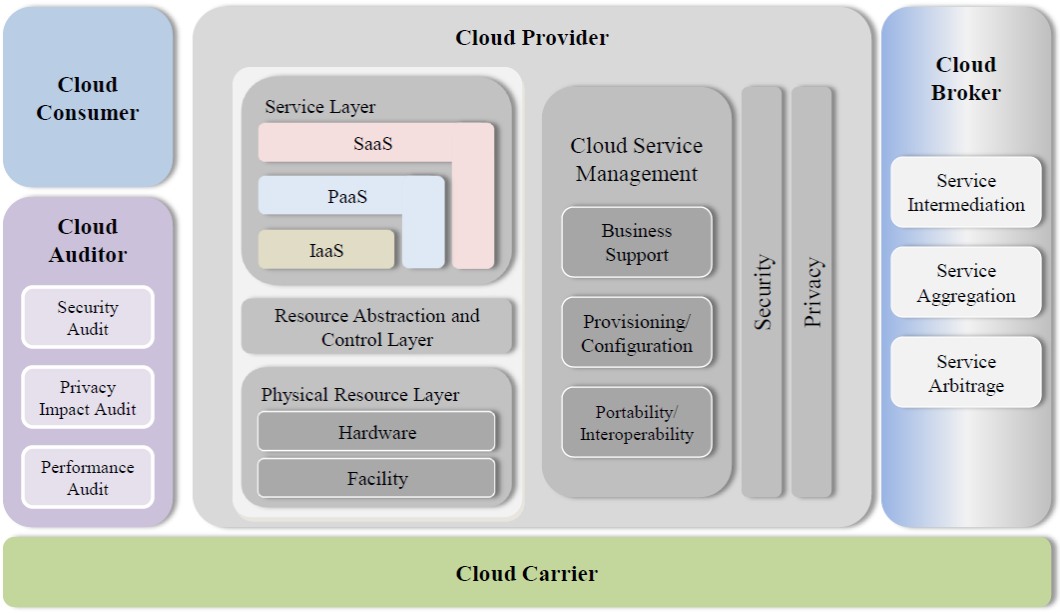


Figure 2.1 NIST cloud computing reference architecture [88]

create complete cloud services within CSP' computational resources for use of potential CSCs, i.e. to develop cloud service components, design cloud service architectures, and implement cloud services provision, etc.

In addition, NIST adds three more parties apart from CSC and CSP, depicted in Figure 2.1. Cloud carrier is the mediator that is responsible for the delivery of cloud services from CSP towards CSC. Cloud broker is the intermediate who manages the relationships and negotiation between CSP and CSC. It can also be assigned to manage the provision and usage of cloud services. Cloud auditor is an entity involved to monitor the use of consumers, or record the performance of CSP for legal purposes.

2.1.4 Cloud Computing Fundament: Virtualisation

In the field of computing, virtualisation refers to a computational resource abstraction technology through which virtual appliances are created from managed computing resources [83]. For instance, OS virtualisation allows to

run another OS within host OS on a single set of physical hardware. A cloud is seen as a pool of virtualised resources from which certain level(s) of service(s) is abstracted based on users' requests [160]. CC is a service-oriented model that relies on virtualisation and distributed computing technologies [89], as depicted in Figure 2.2.

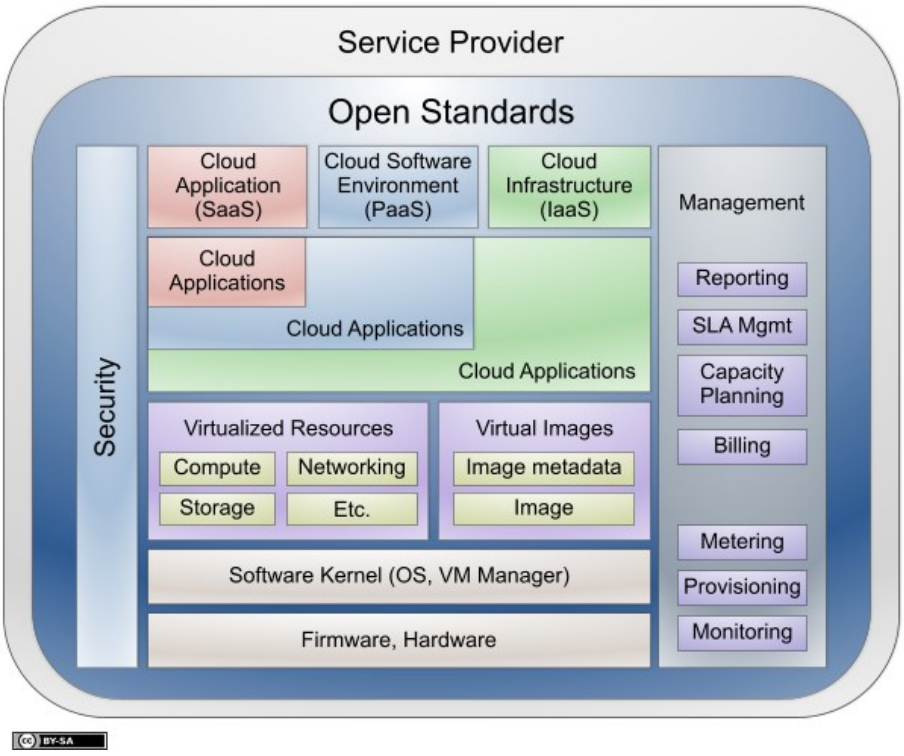


Figure 2.2 Virtualised services provision of cloud providers [67]

Virtualisation technology enables a maximally energy-efficient consumption of physical computer systems, due to the fact that idle hardware operation is minimised [161]. It also assists in distributing workload, e.g. server consolidation is achieved by powering up or shutting down virtual servers based on volume of work. Nevertheless, a number of drawbacks of virtual appliances are discovered [170]: there is inadequate flexibility and adaptability between virtualised appliances and applications. For instance, a user may have to work on different VMs when one tries to use heterogeneous software. Another issue is known as the inefficient use of storage [6]. Although it aims to minimise the idle wastage and unproductive resource consumption, it proves that the preserved storage

overhead is still an issues. As for VM image disk spaces, they are not efficiently consumed.

2.1.5 Typical Characteristics

- *Elasticity*

Elasticity [25] as one of the typical characteristics of cloud services, stands for the ability to scale resource provision up and down rapidly based on real needs of the users. Compared with traditional computing services, it is a distinguished feature as the scaling is rapidly achieved, plus there is no complicated hardware upgrade/downgrade or administration task involved [86].

Elasticity makes CC a “game-changing force for IT” (combined with the on-demand self-service-alike paradigm) [110]. Before this paradigm, elastic IT responds only exist in large-scale organisations which have substantial budgets to develop and maintain the maximum computing infrastructure and software services. Yet, CC offers cost-effective service elasticity that enables very similar IT experience for those with limited funds.

- *Scalability*

Scalability [24] is defined as the ability of to cope with increased or decreased workload through adding or removing system resources based on certain system design. Typically, all systems are considered as finite, so scalability is specified to a certain extent [25]. According to Bondi [20], scalability can be categorised into a series of types: *Load scalability* is regarding the capability of functioning “gracefully”, i.e. no matter at light, moderate, or heavy system load, the system can function without excessive delay or improper resource consumption. *Space scalability* is regarded as the size of memory space can “shrink or expand” but does not grow intolerably depending on real-time system requirement. *Structural scalability*, for a certain system, is seen as the implementation or standards of it can encompass all objects no matter how they grow to some extent.

- *Service Level Agreement*

As cloud services are provisioned by CSPs and are consumed by CSCs, between the two parties, there would always be certain contract(s) which regulate each party's roles, behaviour, activities, etc. The contract is commonly known as SLA [111]. For CSPs, it usually state the duties (e.g. reliability, availability), liabilities (e.g. on-demand, pay-per-use, QoS), compensations, etc. For CSC, there are a series of user agreement to follow and comply. Understanding of cloud services vary from user to user, it is not easy for CSP to produce appropriate SLAs that balance well between technical and general aspects.

In CC, SLAs serves as contract-alike agreements that specify what levels of services are to be provided and consumed between CSPs and CSCs. To a wide extent, it may also involve aspects such as obligations and penalties. Due to their impacts on a cloud service's design, provision, pricing, QoS, considerable research on CC SLA is discovered [38].

- *Reliability and availability*

For the provisioned cloud services and resources, reliability and availability are often guaranteed by the relevant CSPs at a certain level [166]. Typically, cloud applications are regarded more reliable and available than traditional self-maintained computing applications. Fundamentally, this is mainly due to the fact that public CSP usually invest heavily to employ service assurance techniques such as load balancing, live migration, and failover recovery, etc. On the other hand, these are seldom in favour for ordinary users or small organisations [93].

2.1.6 Research Focuses

- *Security*

Currently, the security concerns that are likely being considered by the public are enumerated as [146]: Where is the data stored and who has what level of

access? What are the regulatory requirements and how is audition implemented? What about the long-term viability of CSPs? In addition, since the majority of cloud services runs over the Internet, both CSP and CSC can become victims of those well-known malicious networking attacks, like Denial of Service (DoS) attacks, man-in-the-middle attacks, authentication attacks, etc. [11, 125].

Compared to traditional computing security mechanism deployment, CSC has no initiative control over the security policy and the degree of practice. CSP tends to provide ubiquitous access and operation for utilised resources [62]. While compliance and data privacy laws varying from country to country, data locality issues arise when sensitive data flow from one to another. Since cloud consumers do not store their data locally any more whilst they are managed by the cloud vendors, it is not up to the users what security mechanism is implemented and very few providers can offer security customisability. Similarly, it is often impracticable for CSC to choose networking encryption method over cloud application environment [74].

- *Interoperability*

While many cloud service providers (CSPs) provide unique management portals for their own services and resources, the interfaces, functionalities and service operation environments are mostly diverse. Indeed, this is due to the fact that different CSPs usually offer distinct characteristics for certain service quality of service (QoS), feature, customisability, requirement, etc. aspects [102, 107]. Interoperability is a substantial challenge of CC [128]. Even if many efforts have been made towards CC consolidation and standardisation, various vendors have launched their individual paradigms and services which make the market heterogeneous. The largest gap falls between IaaS clouds, whilst PaaS and SaaS clouds have significantly inadequate flexibility and portability [119].

Generally speaking, the heterogeneity in CC can be categorised into two types [134]: vertical heterogeneity and horizontal heterogeneity. *Vertical heterogeneity*

often exists in different delivery models (among distinct clouds) when services cannot be used in conjunction with each other. Horizontal heterogeneity typically means that data cannot be moved over different clouds despite in the same level of service delivery. Indeed, these gaps usually lead to potential vender lock-in issues and system/process overheads.

- *Service optimisation*

Historically, the efforts made in optimising ICT (Information Communication Technology) energy consumption have been largely focusing on efficient utilisation of physical computational resources e.g. green networking, storage and computation in large scale data centres [153]. In the era of CC, however, green optimisation should involve two sets of major objectives: green service (resource) provision [80] as well as green service (resource) consumption [43]. While the former is largely focused with a diversity of approaches proposed, the latter is seldom adequately addressed.

Statistics shows that large and complex server farms and data centres all over the world constitute the majority of global ICT energy consumption [137, 112]. This attracts several attentions and results into numerous research practices. Addressing the service pool and data centre resources utilisation, the optimisation approaches are seen as resource virtualisation [12], server consolidations [54], workload consolidations [70], dynamic voltage and frequency scaling (DVFS) [39], as well as a series of optimised resource allocation and scheduling techniques. These approaches are typically designed for infrastructure owners, e.g. cloud service providers, so that they can run their own infrastructure efficiently [67]. Yet, these optimisations should seldom be regarded as achieving the ultimate energy efficiency, since they only deal with one side of the problem: the service/resource provision efficiency [106, 165]. Currently, very few approaches try to enable service consumption optimisation from the service consumer perspective. In fact, while considering the full life-cycle of cloud services/resources, the efficiency in relation to how end users utilise the provisioned services/resources also matters significantly.

- *Service search and recommendation*

In recent years, many efforts have been made regarding cloud service performance improvement, service scalability and cloud resources management, whereas user-oriented aspects have been neglected [57]. As a result, lacking concentration of the users significantly drag the development pace of CC. As the number of cloud services continues growing whilst the market becomes increasingly complex, CSCs thus, may need to dig deeply to find the optimal services, by researching on a large number of service descriptions, characteristics, properties, SLAs, etc. Furthermore, regarding the services' features, functionalities, customisability and interoperability, etc., existing CSPs offer a diversity of interfaces, standards, policies and SLA parameters, which result into numerous difficulties in service information retrieval, interpretation and analysis [109, 161]. Consequently, these impose urgent needs and great challenges on the specification and retrieval of cloud services, whereas an effective cloud service recommendation system is in demand for a variety of CSCs.

2.2 Service Modelling Specifications

2.2.1 Semantic Web Services

Towards the promises of service oriented architecture (SOA), web services are delivered to achieve a single aim or integrated goals [21]. Yet, the dynamic composition of web services experiences a series of difficulties, e.g. the goal of the web service is not clear, the protocol is not compatible. Indeed, this is due to SOA systems suffer from interpretability and interoperability issues across the Internet. The semantic Web was first raised by Tim Berners-Lee [14]. The idea of semantically define and describe web services are endorsed by many researchers. It realises the feasibility of interpreting details of web services not only to human, but also to machines themselves. By understanding and communicating between each other, web service discovery and composition tasks can be automated operated even without human intervention [31].

According to Martin and Domingue [94, 95], there are four main elements of the Web Service Modelling Framework (WSMF), known as ontologies, web services, goals, and mediators.

Ontologies are constructed by web service and applying fields experts based on facts and consistent logic. The properly defined concepts, relations, axioms, etc. in ontology are known as the semantic foundation that provides accurate information inference and reasoning between machines and humans.

Web services are designed to achieve a single objective, which is a certain part of the whole aim. They are described semantically so that human and machines are able to interpret their functionality, behaviour, and properties, such as interface, protocol, etc. Once a web service in developed and published, it then can be used and reused as a component interacting with others towards a united goal.

Goals are what users are trying to achieve while consuming web services. For instance, a person uses an online payment service with the goal of making a payment. It usually consists of two parts: requested capability and requested interface.

Mediators are involved to deal with interoperability issues between web services. For example, to adjust interface or protocol mismatch between web services, to construct a new goal by composing differently aimed web services, to configure data or ontological semantics heterogeneity across web services, etc.

Web service modelling semantics can offer many advantages for various usage scenarios. *Firstly,* the functional and non-functional service specifications and definitions would present detailed service functionality and additional information, and particularly benefit porting application horizontally in the service community [143]. S*econdly,* the relevant service data modelling would eliminate many interoperability issues for web service data communication [58]. *Thirdly*, with the enhanced service descriptions various types of service

attributes can be disclosed and interpreted easily by both machines and humans. This would effectively assist service discovery and invocation tasks [77].

2.2.2 Existing Cloud Service Modelling Practices

Ontology expresses a body of knowledge of a certain domain by defining concepts, their relationships and restrictions. It is considered as an explicit specification of conceptualisation [161]. In OWL-based ontologies, with appropriate annotations, not only can it be easily understood by humans, but also it is interpretable by machines. Recently, OWL and a variety of web service ontologies [136] prove that such approach has many superior advantages than UDDI (Universal Description, Discovery and Integration [153]), and WSDL (Web Service Description Language [163]).

Youseff et al. [167] proposes an initial architecture towards their unified cloud ontology. Towards the proposed ontology classifications, a layered hierarchy is being built based on logical definition of CC, seen as “cloud applications”, “cloud software environment”, “cloud software infrastructure”, “software kernel” and “firmware/hardware”. Indeed, all CC services/applications rely on the hardware and firmware stack, on which the software kernel layer implements control and monitor behaviours over the entire physical computational resources. Cloud software infrastructure is provisioned on top of the software kernel management; whereas the many cloud applications are achieved above the cloud software infrastructure layer. Nonetheless, it is not clear whether and how their proposed solution actually relate the above layers one another properly.

Indeed, many existing cloud (service) ontologies only concern about limited cloud concepts with tightly-structured entity relations, such as in [73, 79, 167]. On the other hand, a much better solution is to use an open classification and loosely-coupled ontology structure, since this can comprise as many as relevant entities and their penitential relationships. With such implementation techniques, the diversity of CC concepts, service entities and properties would

no longer be isolated to each other. In addition, by using appropriate ontology reasoning engine, new inferred assertions would present more useful knowledge whenever new data is inserted. Consequently, this ought to construct a more resourceful and meaningful ontology in CC domain.

Another important concern is that none of them considered the unique dynamic characteristics of CC. Different from other fairly “static” knowledge domains, CC comprises dynamic entity aspects, the relationships among them are hard to define or describe due to a series of changing factors. For instance, not all “IaaS has the capability to host PaaS”, even if it is commonly regarded that they do, as operating platforms usually run on top of computing infrastructure. While allocating computing power to CSC, actual service provision changes according to a series of activities such as resource availability, load balancing, automated scheduled services, and also users’ demand.

Since CC is regarded as deriving from a series of computing technology, such as virtualisation, distributed computing, grid computing, potential categories and definitions of CC concepts may raise heterogeneous issues based on different perspectives from different computing research fields. Hence, cloud service ontology with appropriate terms that satisfy a high degree of common understanding across associated computing subject areas is also being expected.

2.2.3 Latest Semantic Specification Language: OWL2 New Features

OWL2 extends OWL by adding new syntax sugar, new constructs, extended datatypes, simple metamodeling and extended annotation capabilities [53].

- *New Syntax Sugar*

DisjointClass – for use of defining a series of classes are pairwise disjointed.

DisjointUnion – for use of specifying a superclass is the union of pairwise disjointed classes. It means the superclass subsumes those disjointed classes,

whereas any member of the superclass can only belong to one of those classes.

NegativeObjectPropertyAssertion / NegativeDataPropertyAssertion – for use of indicating the negation for a given property applied on an individual or a class of individuals.

- *New Constructs*

Self-restriction allows an individual or a class of individuals to relate to itself or themselves with an appropriate property.

Property qualified cardinality restriction enables extended range qualified restrictions to be applied to the object/data property cardinality restrictions.

Object properties can be tagged as *Reflexive, Irreflexive,* and *Asymmetric* properties. *Reflexive* means for a given object property a subject can relate to itself and others at the same time; *Irreflexive* property means the property can only be used to relate a subject to others and not to itself; *Asymmetric* property states an object property is directional between two subjects, it is inconsistent if the two subjects are swapped over.

Disjoint Properties can be stated when a series of properties are pairwise incompatible in the ontology, similarly to Disjoint Classes.

Property Chain Inclusion provides a means to indicate a property is composed by a number of other properties.

Keys allow universal unique key value to be inserted to individuals or classes in ontology, by presenting keys it is much easier to locate subjects within the key propertied class.

- *Extended data type restrictions*

OWL2 presents advanced use of datatype property, seen as *DatatypeRestriction, DatatypeDefination, DataIntersectionOf, DataUnionOf, DataComplementOf*.

- *Simple Metamodeling*

In OWL2 ontology, an entity can be stated as both an individual and a class of that kind of individuals. In some cases, with the same name, an individual can be used as an object property, whilst a class can be used as an object property.

- *Annotation Updates*

Annotations can be flexibly inserted to individuals, classes, properties, axioms, ontology, and annotations using OWL2. They work as annotation assertions, which do not carry OWL2 Direct Semantics and will not be reasoned.

2.3 Cloud Service Search and Recommendation Approaches

2.3.1 Search Engines for Clouds/Cloud Service

In order to assist cloud service search and discovery, Han, Kang and Kim [60, 73, 79] implement a series of research and experiments based on their proposed cloud service ontologies. Their “cloudle” system allows users to input their service requirements through a web portal, and after searching and comparing all recorded cloud services, possible candidates are displayed along with a numbers of parameters (similarity degree, QoS, etc.).

Two separate cloud service ontologies are used for evaluation purposes in above experiments. The first one [60], comprises only basic cloud service concepts with sub-super relationships among them. For example, “DaaS (Data-as-a-Service), SaaS, PaaS, CaaS (Communication-as-a-Service), and IaaS” “is-a” “Cloud System”, which means that “DaaS, SaaS, PaaS, CaaS, IaaS are all subclasses of Cloud System”. Such way of using OWL only has the capability of categorising cloud concepts properly. Yet, there are hardly any obvious

advantages comparing with database techniques, for the reason that complex relationships, description logic, and reasoning are not involved.

The second ontology [79] is developed to deal with advanced queries and comprehensive results and recommendations. Object properties are used to relate class categories with specific relationships, whilst datatype properties are used to point out that some classes fulfil certain datatype restrictions. For example, “has programming language” can be an object property that relate “PaaS” to “Java and C++”; “has memory” of “integer” between “128” and “12800” can be a datatype property that applies to an instance of “IaaS”. In fact, this manner cannot effectively deal with updates occurred in the clouds. Since it allows only fixed axioms to be inserted, the changes to be presented to the ontology may grow exponentially.

The search and recommendation system proposed [73] consists of “query processor, user profiling, similarity reasoning, price and timeslot utilities matching, and rating” components. By inputting requirements and parameters like, type, function, price, time slot, etc. of services, users obtain a list of most applicable service candidates that are similar to what they have entered. The similarity value is calculated based on consulting their cloud service ontology.

Despite the fact that their experiments show that “cloudle” makes some differences, a number of points are to be noticed. Firstly, the ontology used is still not expressive enough to describe comprehensive information of CC concepts and entities. Cloud services are not isolated one another; instead, there are complex relationships among each other. Secondly, to compare service candidates, not only should “hardware” oriented aspects are compared, additional attributes should also be considered, such as SLA, security, dependent restrictions, etc. Thirdly, the web portal and user interface do not seem to be very friendly. In order to assist all types of users, and especially non-expert users, an interpretative mechanism will result in significant differences.

2.3.2 Service Repository and OWL-enabled Applications

Many efforts [9, 59, 95] have been made regarding semantic specification of (web) services. Nevertheless, none of these existing approaches have considered the dynamic aspects of software aspects. Therefore, those service repositories, known as “static asset” repositories, cannot present the adaptive evolving nature of the software assets.

Ontologies are widely used in service repository building, where service specification semantics facilitate service discovery as they are enriched with ontology description languages [129]. In fact, a huge number of service specification techniques as well as service repositories are semantic based. By either using service description enhancements [100, 123], annotating service details [80], or adding protocol information [75], the semantic repository approach has been adopted as a suitable means for service matching.

The semantic web [76] and OWL-based modelling techniques [49] provide feasibilities of identifying, sharing, and reusing data among web applications. They can not only assist human to understand the services behaviours and interfaces, but also allow machines to communicate with each other for relevant application interaction tasks.

As many service providers often follow similar service/resource provision paradigms, the provisioned services and recourses can be specified using ontology semantics with similar modelling style [128]. Indeed, for CC domain, ontological modelling approach can be utilised to formally describe a wide range of CC entities, concepts, attributes and relations. Therefore, the generic specifications can effectively addresses interoperability issues among heterogeneous clouds [134]. Semantic-based cloud (service) ontologies are hence considered more expressive than other specification models. Moreover, another benefit offered by ontology modelling is known as reasoning. It ensures the (specification) data consistency within the ontology whilst additional inferred information may be produced whenever changes are made to it.

Deng et al. [36] proposes an enterprise service catalogue management framework using ontology oriented approach. The declared ontology clearly presents information of the services and the involved processes based on detailed analysis of the common user requirements and technical aspects. Apart from above, with additional algorithm support, the ontology representation allows optimised selection and combination of services according to complex requests. The limitation of such approach, however, is that concepts similarity judgment remains an issue across heterogeneous ontologies [161].

2.4 Dealing with Uncertainties for Cloud Computing

2.4.1 Theory Support for OWL Fuzzy Extension

In fact, unlike web services and many other domains, CC involves a variety of vague and imprecise descriptions, terms, categorisations, etc. This can result into various specification issues. For instance, according to the majority of literature, “availability” and “security” are two separate service properties, yet some [68] argue that availability is a sub category of security; for those diverse service types and characteristics, should Amazon S3, Dropbox and Google Drive be regarded as SaaS, PaaS, IaaS or Storage-as-a-Service? Do they have the same extent (degree to the capability) towards scalability, reliability, interpretability? Indeed, conventional OWL/OWL2 modelling techniques cannot handle the above scenarios effectively. Originally, they are designed to clarify crisp knowledge with concrete axioms, either true or false. Fundamentally, this is due to the formal description logical (DL) consistency requirement which does not support such fuzziness [19, 42].

Fuzzy logic [168] (FL) is a well-known extension to DL that has been used widely in many fields for decades. It includes two sets of theories: fuzzy set theory describes vague subsumption between a class and its members; fuzzy relationship theory [124] specifies uncertain relationships between individuals and classes.

2.4.2 Fuzzy Logic Theories

- *Fuzzy Set and Membership*

In crisp set scenario, an individual element can either belong or not belong to a certain collection, based on the fact whether the individual complies with the characteristics of the collection. For example, an apple is obvious an instance of fruit; a cucumber is not an animal. Yet, fuzziness and vagueness exist widely around us, e.g. dark colours, big cakes. To address these and similar kind of uncertainties, fuzzy logic was introduced. According to FL theory [167], a fuzzy set is known as: a collection that has fuzzy characteristics or a class that is imprecisely defined. Moreover, to indicate a subject is or is not an instance of a fuzzy collection, the float μ is typically used as the truth rate of the unit interval [0, 1].

A membership degree is, thus, defined as the degree to which an individual is considered to be the instance of a class. Value of interval (0, 0.5] means “the statement is less likely to be true” and [0.5, 1) means “the statement is more likely to be true”. Assuming an individual x and two fuzzy sets A and B that it may belong to: $\mu_A(x) = 0.9$ stands for x is very likely to be the instance of A; $\mu_B(x) = 0.2$ stands for x is very unlikely to be the instance of B. In addition, they satisfy

$$\mu_{A \cap B}(x) = \mu_A(x) \wedge \mu_B(x) = \min\{\mu_A(x), \mu_B(x)\} = \mu_B(x) = 0.2$$

$$\mu_{A \cup B}(x) = \mu_A(x) \vee \mu_B(x) = \max\{\mu_A(x), \mu_B(x)\} = \mu_A(x) = 0.9$$

It means that the degree of x belonging to the intersection of A and B is the minimum μ of $\mu_A(x)$ and $\mu_B(x)$, which is 0.2; and the degree of x belonging to the union of A and B is the maximum μ of $\mu_A(x)$ and $\mu_B(x)$, seen as 0.9.

- *Fuzzy Relations*

Crisp relations between subjects are known as one subject is completely related or unrelated with another subject over certain named relationships. For instance, a mother relates her child with the “give birth to” relation. In a slightly complicated case, a single subject can relate to a set of subjects with the same relation in the same time. Also, there can be a relation, with which each subjects of one set relates to every individual of another set respectively. For example, A = {a_1, a_2, a_3}; B = {b_1, b_2, b_3}; the relations between A and B denote:

$A \times B = \{ a_1 \times b_1, a_1 \times b_2, a_1 \times b_3, a_2 \times b_1, a_2 \times b_2, a_2 \times b_3, a_3 \times b_1, a_3 \times b_2, a_3 \times b_3 \}$

In the fuzzy relationship [124] theory, strengths can be allocated on top of crisp relations between ordered pairs of two collections. For example, to express that a father knows his son better than his wife does, two strengths can be used along with the two relations: the father knows his son at the degree of 0.9; the mother knows her son at the degree of 0.8. Therefore, for a fuzzy relation, strength can be asserted to express applicable or constraint degree.

A fuzzy relation R over two sets U and V denotes: R: $U \times V \rightarrow [0, 1]$; when R = 1, it means there is a crisp relationship R over U and V, and R = 0 means the negation of R = 1. Additionally, each relation r_{ij} between ordered pairs of U and V can be displayed in the matrix of $R(r_{ij})_{u \times v}$ (where $j \neq k$ and $u \neq n$). For instance, say, U = {u1, u2, u3}, V = {v1, v2, v3, v4}, $R_{U \times V}$ denotes:

$$R_{U \times V} = \begin{bmatrix} u1 \times v1 & u2 \times v1 & u3 \times v1 \\ u1 \times v2 & u2 \times v2 & u3 \times v2 \\ u1 \times v3 & u2 \times v3 & u3 \times v3 \\ u1 \times v4 & u2 \times v4 & u3 \times v4 \end{bmatrix}$$

2.4.3 Fuzzy OWL Extensions

The necessity of fuzzy support in semantic web has been widely agreed [16, 17, 91, 141]. In their work, FL and fuzzy DL reasoning are introduced to OWL (1&2) to cope with fuzziness occurred in domains of ontology. Stoilos et al. [140, 141]

suggests an achievable approach by using new syntax (like "*owlx:degree*") as OWL extensions; Bobillo and Straccia [17] initially advocates wrapping fuzzy theory with OWL individual/class concepts for fuzzy expressions, then recommends using annotation itself to present the fuzziness afterwards [16, 17].

- *fuzzyowl2*

An early work on fuzzy support of OWL2 is proposed in [19], where several fuzzy concepts are introduced to OWL2 ontology in the form of OWL classes and individuals. Yet, it is proved that their approach is not applicable for the entire fuzzy theories in many cases. Not only the way fuzzy theory is adopted caused logic inconsistencies, it also accelerates the growth of the ontology exponentially.

- *SWRL-F*

Another OWL2 fuzzy extension is known as Semantic Web Rule Language – Fuzzy (SWRL-F) [162]. The approach does not use extra built-in either. Instead, OWL OP and individuals are used as key factors to construct fuzzy assertions. The implementation does not affect the consistency of the applied ontology. Yet, general limitation of SWRLJessTab and fuzzy inferences are limited based on the logic rules of consistent OWL DL.

- *fowl*

Annotation based "fowl" alike fuzzy support is proposed in [139], which against their previous work. By using solely customised OWL annotations to express the fuzziness, quite comprehensive fuzzy theory is represented in their proposed <fuzzyOwl2>. By using their user friendly Protégé plug-in, modification of ontology seems to be easy. In addition, fuzzyDL [17] and DeLorean [18] reasoners are also developed in order to support fuzzy reasoning. Despite the fact that it shows optimised support of complex fuzzy concepts and theory, the way it is applied into OWL2 is a bit controversial. The

fuzziness is applied in the annotations, which is hardly a genuine OWL2 support; it does not support traditional DL reasoner, either.

- *f-OWL*

New syntax based extension is recommended by [140, 141], who arguing to support f-OWL, primitive syntax shall be extended, such as by adding owlx:degree, owlx:ineq, etc. However, after many years, there is not much development and their syntax family is far from completeness. Not to say the design is not entirely OWL2 focused as well.

2.5 Summary

Existing cloud (service) search engines cannot effectively understand the constraints and dependencies among resources within the same cloud or across multiple clouds, whereas none of them comprises enough information to reveal the various types, functions, and features of cloud services. This results into significant limitations for search and recommendation tasks. Moreover, current cloud search and recommendation systems cannot efficiently deal with the frequent updates occurred in the clouds along with the evolvement of the cloud services. As a consequence, the existing systems/tools would eventually, decay as CC evolves progressively.

The efforts on building CC domain ontology can be traced back since 2008. This proves that there is substantial necessity of such. Yet, for the existing cloud ontologies, they are seldom comprehensive enough to capture the wide range of unique characteristics of cloud services, i.e. elasticity, scalability, reliability, security, interoperability, SLA, etc. None of the existing cloud domain ontologies is built upon or able to reveal the fundamental aspects of it.

Fuzzy extension to OWL has been a widely discussed topic in Appropriate Reasoning and Fuzzy Systems fields. Although distinct solutions are proposed, none of them is free of limitation: As a few tend to use new syntax to represent the fuzziness, their modified ontologies fail to work with all existing OWL-

enabled applications. While some wrap the whole fuzzy theory into OWL annotations, they are doubt whether that is a genuine OWL supported approach.

Process-based service modelling mechanism exhibits superior characteristics of assisting search and comparison tasks, yet suffers from exponential elements growth and effort consuming pre-design in extreme cases. In the meantime, dynamic service repository system offers a variety of advantages while tackling software aspects evolvement, but it needs an efficient way of processing entire ontology entities.

In the meantime, relevant literature regarding the proposed approach is explored. While exploring virtualisation and semantic web service, possible solutions toward the above issues are developed, i.e. to develop a semantic cloud service specification framework that is capable of assisting a combination of functions including cloud service search, recommendation, retrieval, management and potentially comparison, evaluation and orchestration. Within the framework, some cloud service ontologies will be developed. They would comprehensively specify a diversity of cloud aspects and entities. As all of such entities and aspects are to be properly addressed and related according to appropriate dependencies and constraints among each other, this then spreads across additional research areas and rationales (e.g. OWL2 new features, process-based service modelling, and fuzzy logic theories). By detailing how they work, the mandatory elements of the proposed approach have been illustrated.

Chapter 3 Related Work

This chapter discusses the related work closely relevant to the proposed ontologies, approaches and cloud service assistance tool. Firstly, the existing semantic cloud (service) specification models and the current practices of cloud service recommendation system are described. Subsequently, the work on fuzzy ontology extensions is talked. Finally, the research on open cloud service and resource specification, API and remote service management tool is reviewed.

3.1 Cloud (Service) Specification Models and Recommendation Systems

3.1.1 Ontology-based Cloud Computing/Service Knowledge Representation

Historically, cloud (service) semantic modelling research has involved various ontological approaches such as single ontology [144], multiple-layered ontologies [131] and multiple ontologies [93], etc. The semantic platform for cloud service annotation and retrieval [122] utilises multiple ontologies of different domains. Being advanced in its annotation term extraction and indexing techniques plus the integrated ontology evolution module, it can implement ontology updates according to the service concept information found on Wikipedia. In their incremental work [121], GATE [30] is employed for automatic service annotation and ontology evolution. Nonetheless, annotation specification, parsing and retrieval are a basic use case in ontology modelling. Such updates in annotations would not drive sound ontology evolutions, i.e. generating new inferred knowledge.

Alternatively, other work (e.g. [99, 161]) does employ class, object property (OP), data property (DP), assertions as well as basic inheritance and inference, etc. in their ontologies. Nevertheless, most of the ontologies are primarily designed to work for certain limited service categories: e.g. infrastructure

services [60, 73, 93, 161, 169], platform services [96, 144], software services [121, 122]. FCFA [96], for instance, is a hierarchical federated resource exploration and sharing framework which drives federated cloud cooperation and eliminates interoperability issues among independent organisations and providers. The proposed ontology only concentrates on the relationships between organisations, communities in terms of federation contracts, SLA agreements, plus the various physical and virtual resource properties and parameters involved. CoCoOn [161] is an infrastructure service ontology which comprises both functional and non-functional specifications of cloud VM and storage resource aspects; it still does not involve service information across wider resource abstraction levels. Although Cloud Ontology [73] is able to specify service information of a variety of cloud services, it only discloses some basic aspects regarding the diverse service functions/levels. In fact, for almost all existing ontologies, the cloud service and CC concept specifications are not established evenly across multiple abstraction levels and service function categories. Indeed, except mosaic [7], none of other ontologies reveals any explicit details regarding the many service functional, non-functional properties or relationships. Besides, there is no other that attempts to specify the various service agility aspects or the most appropriate specifications through fuzzy extensions; none of current practices supports collaborative editing for the modelled service specifications.

3.1.2 Cloud Service Recommendation Systems

Existing service recommendation/discovery systems/tools are seen limited in terms of their overall applicability, flexibility and comprehensiveness. Some [60, 161] are found focusing on IaaS-centric service recommendation. Specifically, CSDS 60 presents an example of discovering VM services according to search parameters such as virtual CPU architecture/frequency, memory/storage size, network parameter, operating system (OS), etc. CloudRecommender [161] offers enhanced functions which accept both functional and non-functional service properties as recommendation requirements. Nonetheless, due to their limited service category applicability, the two systems cannot facilitate

comprehensive service recommendation in a wider domain (with inclusion of PaaS and SaaS). Differently, the cloud repository and discovery framework [144] advocates recommending cloud services from a business and cloud service combined ontology. However, since the recommendation is implemented through querying business-relevant service properties, it implies that the recommendation process would be excessively business–focused. Cloudle [73] can produce a list of discovered services along with their similarity values from several services types by offering diverse search criteria and options of, e.g. cost, time, function, technical requirements, etc. Yet, the similarity computation relies on purely numerical service properties and, therefore it still cannot effectively handle comprehensive service specification. On the other hand, non-ontology-based service recommendation system, like the collaborative service recommender mechanism [151], is an alternative that specifically deals with service matchmaking through consumer rated service qualities against users' profiles. Yet due to the prototype mostly concentrated on non-functional service aspects (e.g. response time, availability, price, etc.), the limited functional requirement processing capability would result into poor overall service recommendation.

In summary, currently there is not a comprehensive means of cloud service search, retrieval and recommendation which covers a diversity of service/application domains, whereas none existing tool attempts to involve search/recommendation requirements regarding any details regarding the unique (agility) aspects of cloud services, e.g. scalability, adaptability, interoperability, etc.

3.2 Ontology Fuzzy Extensions

On the basis of fuzzy theories (described in Chapter 2), a series of fuzzy extension techniques propagate. FuzzyOWL2Ontology [19] advocates a merging approach to import the fuzzy representations, which are wrapped as ontology entities, to the target ontology for fuzziness expression. The drawback is known to be its limited support of complicated fuzzy scenarios as well as the

considerable extra overhead. In contrast, new syntax-based fuzzy extension [139] is proposed where the primitive OWL2 syntax is extended with “owlx:degree”, “owlx:ineqType”, etc. elements. Nevertheless, due to without additional extension mechanism support (for fuzzy assertion and interpretation), such modification would have little compatibility with current main stream OWL/OWL2 tools and can only be implemented and interpreted manually by humans. The annotation-based fuzzy extension [16] presents another approach, seen as to place the fuzziness in OWL2 annotations. With comprehensive fuzzy set and relation theory support using “fowl” and “fuzzyOWL2” syntax, a Protégé [62] plug-in is also developed for easy fuzzy modification and illustration. Yet, applying fuzziness in annotation property would suggest that such extensions only present some fuzzy annotation descriptions for the ontology entities whilst they do not influence the individuals, classes or their relations in the ontology in any means concretely, i.e. such fuzziness assertions do not present genuine facts of them. While all the above approaches remain unideal, the OWL2 natively supported fuzzy extension [42] demonstrates a promising technique by using fuzzy tag-alike modifications. The extension employs no further new syntax but only OWL2 DP assertions, which brings a series of advantages: the fuzzy extended ontology is readable by all mainstream OWL2 tools (like Protégé) whereas traditional DL reasoners like FaCT++ [152] and HermiT [131] are supported; Changes made to the asserted fuzziness can trigger ontology inference changes.

In spite of the above FL-based fuzzy extension approaches, recently, probabilistic logic network [51] (PLN) is raised and known as another complete systematic and pragmatic knowledge representation theory specifically developed for uncertainness assertions and inferences. In comparison with FL, it extends the fuzzy set and relationship theories and their reasoning applicability to a great extent with complementary rules, strength formulas and inferential truth value equations using extended formal notations, e.g. it differentiates FL’s fuzzy membership theory into a number of detailed scenarios (e.g. degreed belonging, chanced belonging, sharing partial properties and

overall weighted judgment); it fulfils FL's relationship theory with higher-order and N-ary logical relationships. With these advanced theoretical support, this thesis advocates the extended version of the OWL2 natively supported fuzzy extension approach [42] for ontology fuzzy axioms revealing and handling.

3.3 Toward Unified Cloud Service/Resource Specification and Management

In the last decades, considerable efforts have been made on enhancing the interoperability and portability of cloud services. The practices are widely discovered in open cloud API developments, comprehensive service/resource specification frameworks, unified cloud management protocols/drivers, etc.

3.3.1 Open Cloud Service Specification Framework

The Open Cloud Computing Interface [107] (OCCI) is one of the earliest practices in the field. Originally, it was developed only to deal with IaaS service remote management tasks such as resource deployment, monitoring and automated scheduling. Yet later, the evolved Rendering and Extension specification frameworks on top of the Core Model enable a much wider application for PaaS and SaaS services, which consequently make it a generic management API for a diversity of cloud resources.

The OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA [150]) is a recently established standard for clouds. With the aim to enhance cloud service portability, it enables specifications for diverse cloud service resources, their relationships and operational behaviours. With several templates (e.g. service/policy templates) and types (e.g. node/relationship /requirement/capability types) specifications, the topology framework can provide semantic support for many cloud service management and orchestration tasks.

Other than the above well-established practices, a series of research projects are also implemented towards the aim. mOSAIC [7], for instance, advocates a

so-called application/provider/language-independent for service semantic specification. Resting on the mOSAIC ontology as the knowledge base for semantic resource discovery, it allows separation of application-logic and cloud layers while enabling service portability. Likewise, another work targeting at enhancing the interoperability of cloud services is seen the RASIC framework [90]. It is composed from three horizontal layers (i.e. service frontend, SOA, virtualisation/execution) and two vertical layers (i.e. semantic and governance). Similarly, the Intercloud [35] architecture comprises multi-layer cloud service models and a series of management, federation and operations frameworks. They serve as cloud middleware to support the service integration. However, these approaches are developed mainly for infrastructure services (resources). This limits the application towards wider service domains/categories.

3.3.2 Open Cloud Service API

In the meantime, a number of language-dependant Cloud APIs are also discovered. Compared with the above ones where either service specification, protocols or management portal are also available, these are only stand-alone libraries, which are used for cloud service/resource management via a generic API.

Deltacloud [34] is provides an abstraction API that enables service management functions for a number of IaaS resources. The wide range of CSP support makes it feasible to manage heterogeneous resources across diverse clouds. Fundamentally, it runs a series of drivers serving as individual service adapters for each CSP specifically. Each driver would serve as the latest native API associated with its own CSP. Consequently, Deltacloud API along with the management interface enables long-term stability for cloud resource utilisation. This means users would no long need to worry about the differences while handling services across distinct clouds, nor the compatibility issues incurred for frequent version updates.

Libcloud [5], for instance, is a Python library that offers wide supports for more than 30 popular cloud service providers. The library provides four main categories of interfacing functions: compute, storage, load balancer, DNS. In addition, Fog [47] is known as the API library for Ruby developers, which offers similar functionalities. The library has flexible support for a variety of services from mainstream CSPs. Jclouds [4], on the other hand, is a java API library that supports a wide range of existing CSPs. It can be applied for various cloud service categories and purposes for IaaS compute, platform, database, storage, etc. services. Similarly, Dasein cloud API [146] is another example of Java-based cloud service interface. While aiming to eliminate the interoperability issues and enhance the efficiency while building cloud applications using multiple CSP resources, it offers adequate supports for a diversity of Clouds and service platforms.

In addition to the above industry projects, some open cloud service API research is also found in the field. Bastião Silva et al. [8] propose a common API for delivering services over multi-vendor cloud resources, where SDCP (service delivery cloud platform) is presented. Basically, the platform models the diverse cloud entities (e.g. agent, domain, and provider) and manages cloud service data and abstraction (streams) conventions; the cloud controller component provides interfacing functionalities such as provider credential aggregation and service resource access, through the cloud gateway which loads new cloud services and grants authentications. In addition, Petcu et al. [113] proposes the mOSAIC java API as an example of open interface for service deployment and portability. Similarly, in [90] another design of open cloud API is illustrated. However, a known drawback of the approach is that it would easily fail to deal with the uniqueness of similar services for their advanced or newly updated features, due to the fundamental nature of preserving the maximum common aspects for them.

3.3.3 Service/Resource Management Tools for Heterogeneous Clouds

Bernabe et al. [13] demonstrates an access control system for multi-vender cloud resource management. Using ontological modelling techniques, the proposed ontology handles the specifications of the various entities (e.g. cloud, system, software, etc.) involved, whereas the authorisation model deals with the roles, identities and privileges aspects for authorisation and authentication tasks. Although the approach is advanced for its comprehensive support for both conditional and hierarchical role-based access control, the application is currently limited to AWS EC2 resources.

For the aim of a unified cloud storage acquisition, Cloud Data Imager [45] (CDI) is proposed as a complete system to provide comprehensive functionalities for access and managing storage resources across diverse clouds (i.e. Dropbox, Google Drive, Microsoft SkyDrive). The developed CDI library is able to handle a variety of functions including user authentication, folder listing, file downloading, etc. Another work addressing resource utilisation monitoring issues over heterogeneous multi-tenant clouds is found in [116]. The work proposes DARGOS architecture which can provide highly reliable and scaling monitoring functions, where insignificant overhead is resulted. Despite their advantages on dealing with the respected cloud service tasks, the above systems and approaches are restricted by their fundamental sole usage design and would only work for limited cloud service models/types.

A model-based cloud service integration platform is advocated 84 to drive service orchestration for business purpose. The proposed framework tackles the issues by looking into three levels of modelling: cross-organisational business processes modelling, service operation/orchestration modelling, and dynamic member services binding modelling. By using the cloud service API encapsulation method, it is then able to interconnect different services and resources as needed, according certain business process flows.

3.4 Summary

Existing cloud (service) ontologies are often based on unbalanced and incomprehensive service and concept specification establishment. For most of them, explicit details regarding services' characteristics, properties and relationships are missing. Moreover, no existing ontology involves the specification and presentation of cloud service fuzziness. Consequently, they have various limitations in terms of the comprehensiveness and depth of the knowledge represented; particularly, they fail to deal with service agility across the abstraction levels and the service categories. These issues prevent current service recommendation systems from providing the most effective cloud service recommendation functions. In fact, fundamentally, this is very likely caused by the conventional inflexible design accompanied by the DL-consistent nature of OWL ontology. From a range of proposed FL-based ontology fuzzy extensions, the new PLN-based OWL2 natively supported fuzzy extension is adopted to develop the loosely-coupled and agility-oriented cloud service ontology. As the fuzziness is imported in a collaborative manner, the proposed approach ought to drive comprehensive and flexible service search, retrieval and recommendation.

In the meantime, there are considerable third-party open cloud service API libraries which are mature and available for use, whereas the majority of them offer wide supports for most popular CSPs and all service categories. Not to say, several large scale CSPs also provide their own native API for public, which are often more efficient and stable. These brings many advantages for CSCs in terms of avoiding vender lock-in, more flexible service/resource management, advanced service usages such as service orchestration and adaptation. In the meantime, despite various service modelling, specification, integration, etc. approaches being proposed, currently none of them can work effectively while handling diverse cloud service categories/types for a variety of tasks: I) a unified management portal for diverse cloud service access and manipulation regardless of the service layer/category/provider/resource (type), II) an interface that allows automated flexible service (operation) orchestration through its built-

in task scheduler, III) a system which can reason the relationships of cloud services/resources and then produce candidate operation process chains for potential service interactions. Consequently, the gaps discovered in existing works significantly limit the effectiveness and efficiency for cloud service management and composition tasks.

Chapter 4 AoFeCSO

- ***Agility-oriented and Fuzziness-embedded Cloud Service Ontology***

In this chapter, the agility-oriented and fuzziness-embedded cloud service ontology namely AoFeCSO is proposed. Toward the first objective, it is designed to comprehensively specify the various descriptions, characteristics, features properties, etc. concerned with CC and cloud service entities. AoFeCSO utilises the latest OWL2 modelling language and incorporates a range of specification assertions for optimal information presentation. In particular, Section 4.1 firstly illustrates the overall ontology foundation design. This then leads to the details of ontology implementation, including relevant object property, data type property and annotation property specification assertions. In addition, as such traditional modelling techniques cannot handle the vagueness and uncertainty appeared in the specifications, an OWL2 fuzzy extension approach in developed. Section 4.2 presents the design of the extension as well as an OWL2 fuzzy specification management mechanism for fuzzy cloud service specification. With the above featured modelling techniques, AoFeCSO ought to serve effectively for various cloud service search, recommendation and retrieval tasks.

4.1 Overall Ontology Design and Implementation

4.1.1 Loosely-coupled Foundation

AoFeCSO is deployed with a “loosely-couple” ontology foundation: it adopts flexible membership classifications, which enables loose (class) boundary restrictions; it follows the reasoning ontology design patter (ReasoningOP [50]), as it maximally utilises property specifications for enhanced reasoning application. More specifically, they are represented as follows:

1) In AoFeCSO, cloud services are asserted as individuals that belong to the respected cloud company classes (instead of belonging to certain service

delivery models). Among those who are related, there are appropriate relationships such as “rely resource of”, “have control over” and “can orchestrate with”.

2) The cloud services delivery model, deployment model, role, party, feature and function specifications are revealed through object relationships. Object property specifications are asserted from a cloud service towards its respected service model/role classes, e.g. EC2 “is delivered as” IaaS; EC2 “is deployed as” public cloud; Amazon “is recognised as” CSP. In this way, in AoFeCSO, a service can have multiple models and roles, in case that the service is uncertainly regarded as both IaaS and SaaS, both public and private cloud, or both CSC and CSP at the same time.

3) The characteristics and properties that cloud services apply are illustrated as they have detailed relationships with the subclasses of the main service attributes, e.g. service characteristics (elasticity, adaptability, reliability, etc.) and service features (monitoring, notification, multiple OS and programming language support, migration and transition support, etc.).

4) In AoFeCSO, except of the main designed service function(s), cloud services are specified to have more functions as long as they can serve the purpose. For instance, IaaS compute services may also provide application development platform, network, database, or storage functions.

4.1.2 Agility-oriented Design

In the field of CC, agility is generally referred as the ability of a cloud service to react appropriately and rapidly to a series of requirements such as adaptation, customisability, interoperability [67]. In fact, such reaction capability would most likely count on a diversity of fundamental service elements, including solid service design, flexible resource provision, comprehensive monitoring, notification, security, backup and orchestration supports, etc.

Fundamentally, the functions a service can achieve should matter the most regarding its agility, since different functions require distinct architecture designs and resource provisions [67, 92]. Most SaaS services, for instance, rely on fairly limited computational resources and provide single or few limited functions. Meanwhile, typical PaaS services do not have fixed application-scale functions, and instead, can be used to develop or deploy a variety of applications/services where various (potential) usage/functionalities can be achieved. Similarly, for those IaaS services which are designed for general computing needs, they often offer greater service control, access and customisation in both functional and non-functional aspects whilst they can be used to achieve even more (potential) usage purposes. Indeed, the ranges of functions and resources a service is deployed decide how agile it would act during service composition, whereas agility inevitably becomes the link while specifying the above service function aspects and their potential interactions.

The various characteristics and features of cloud services can be seen as a series of further information regards their main and potential service functions [69, 92]. Elasticity and scalability, for instance, are two typical cloud service characteristics. While describing their sub-concepts such as available VM sizes and scaling options plus the further details of vCPU clock speed/cores, intranet/Internet connection speed, memory and virtual storage sizes, etc., all of these aspects are extremely relevant to service' agility as they are proofs detailing certain service' capability of scaling either up/down or in/out as required. Likewise, platform, OS, programming language and application programming interface (API) supports are evidences of agility. The lists of available platforms, OSs, programming languages, APIs are facts that state a service's interoperability and configurability. Similarly, detailed notification, monitoring and security aspects are seen relevant to agility. Notification basically comprises the different service usage notifications and various service health notifications. In general, monitoring consists of a diversity of service element notification, log monitoring, performance monitoring, and security monitoring. Security is generally divided into access control and data security:

access control comprises the different layers that a cloud service supports for its security implementation, e.g. application layer, data layer, network layer, process layer and system layer [154, 156]; data security outlines the data encryption and management supports for its security implementation, e.g. client/application encryption, data loss prevention, database encryption, externally managed encryption, file/folder encryption and digital rights management, instance managed encryption, link network encryption, and provider managed encryption, proxy encryption [157, 158]. Indeed, all these aspects above are often deployed as the guarantee for agility requirements, since they ensure the availability, reliability, integrity, confidentiality of the various agility responses. Consequently, the above service aspects become the detailed reflection of a service's agility.

As a result, as Figure 4.1 illustrates, agility becomes the bridging aspect that incorporates cloud service functions, characteristics and features, both functional and non-functional. To this extent, agility is then seen as the overall reflection of a cloud service's profile data. This is how AoFeCSO models cloud service specification by focusing the in-depth cloud service concept details and their relationships (detailed ontology entities are available in Appendix B).

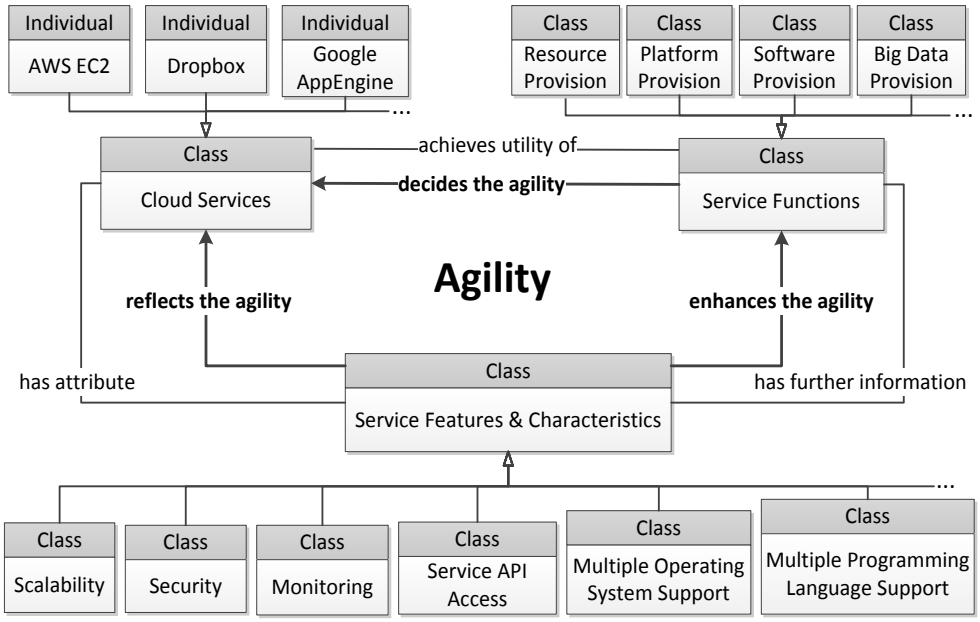


Figure 4.1 Agility-oriented Ontology Design

4.1.3 Ontology Construction

Built on the ground of the existing cloud (service) model knowledge, AoFeCSO adopts full range of OWL2 property assertions, where several different property handling techniques are employed. Figure 4.2, Figure 4.3 and Figure 4.4 demonstrate the extensions AoFeCSO achieved in contrast to other existing models (i.e. [73, 93, 122, 144, 159, 161]).

4.1.3.1 In-depth Assertion of Cloud Service Object Properties

In ontology, an OP declares a certain relationship between two entities. While existing practices [93, 144] utilise OP for attributing cloud service characteristics, functional and non-functional properties, very few touches the details of how or how well those cloud services own these characteristics and properties.

Shown in Figure 4.2, AoFeCSO describes lower-level details regarding the service characteristics and features. For instance, scalability is divided into vertical scalability and horizontal scalability, where each of them has individual sets of concepts. Security comprises access control and data security; each category leads to own sets of security aspects [3, 157]. By digging into the details and relating them with appropriate cloud services, AoFeCSO is capable of expressing in-depth facts of cloud services' characteristics, features and functionalities.

4.1.3.2 Explicit Assertion of Cloud Service and Concept Relationships

Many cloud companies and providers have certain industry relationships among each other. Meantime, several cloud services are built with the ability to interact agilely with others, i.e. they have adaptability and interoperability by nature and hence can be composed towards customised/enhanced functions. Besides, there are obvious/hidden relationships among a variety of CC concepts such as service characteristics, features and functionalities, e.g. scalability is often

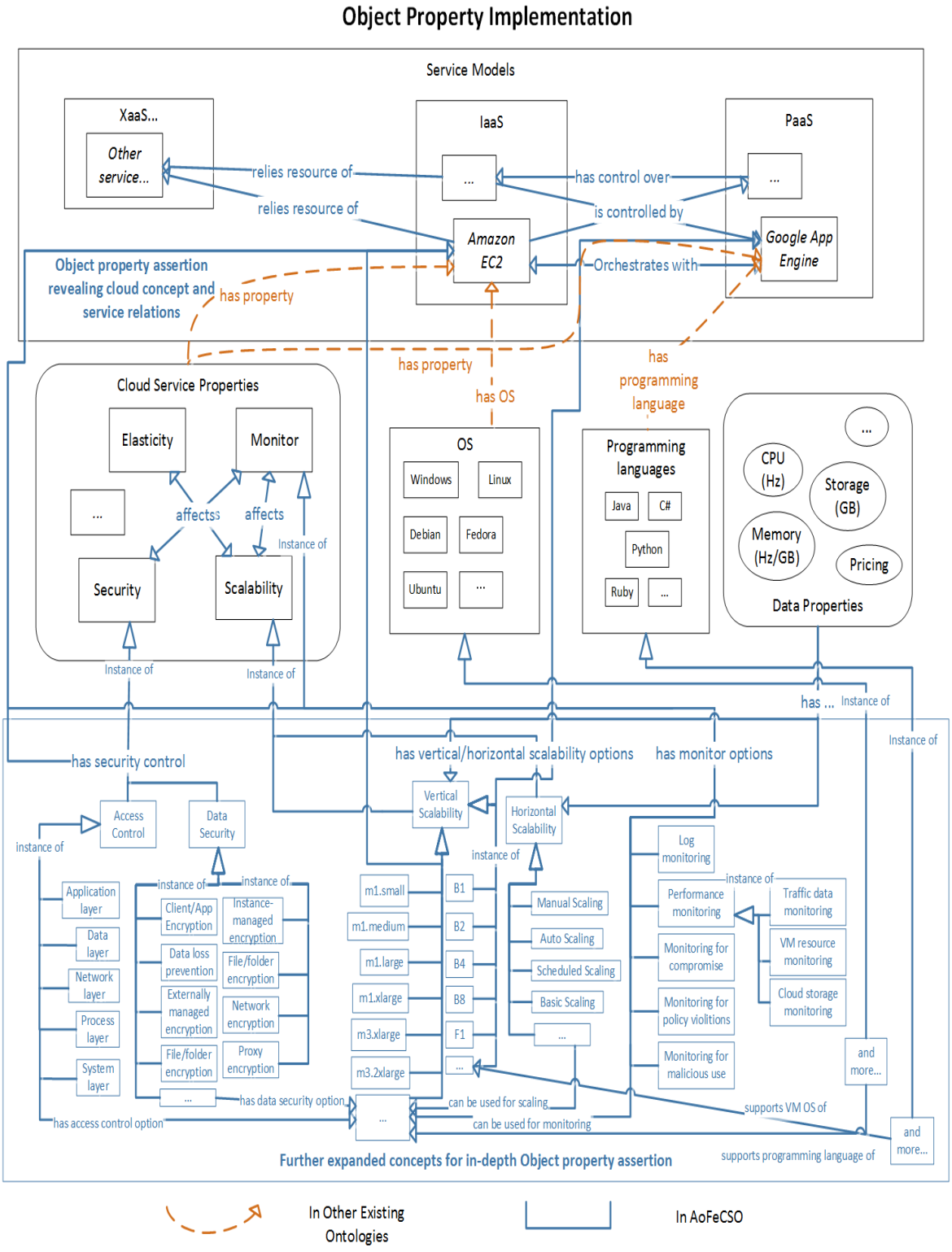


Figure 4.2 Advances of AoFeCSO in Dealing with Object Properties

attributed to elasticity to certain extent; monitor parameters can affect services' scaling and load balancing behaviours. However, for the majority of the existing models, such interoperability and concepts relations are not explicitly addressed and expressed [7, 60, 73, 93, 122, 144, 161].

In contrast, demonstrated in Figure 4.2, AoFeCSO covers these aspects in the form of individual-to-individual, class-to-class and individual-to-class OP assertions among cloud services, companies and other concepts. According to relevant information sources, the various direct/indirect and strong/weak relationships are explicitly revealed using, e.g. "has industry relationship with", "is controlled by", "can affect", etc. Furthermore, such OPs are also asserted with property characteristics such as "transitive", "symmetric" and "inverse property of", which allows DL reasoner to reason new inferred axioms. In this way, AoFeCSO becomes a densely interconnected ontology in which very few concepts are seen "alone" on its own.

4.1.3.3 Categorised and Comprehensive Assertion of Cloud Entity Data Properties

Most existing ontologies solely or largely focus on clarifying the numerical data attributes of compute cloud services [73, 93, 161]. In contrast, AoFeCSO employs DPs for much wider specification usages. As illustrated in Figure 4.3, it employs diverse DP types, such as String, Boolean, Data time, etc. According to cloud services' delivery models, the DPs are divided into sub categories. For instance, IaaS compute services have "vCPU core, frequency, memory size, network performance", etc. PaaS application platform services have "programming language version support, maximum size of application file, maximum total number of file per directory", etc. SaaS file storage services have "binary difference support, file session support, individual size limit, revision history support", etc.

In addition, cloud service SLA data is specified with DP assertions. This involves specifications of SLA descriptions, obligations and other relevant terms

and conditions, such as “SLA effective date, service commitment, service compensation, service error rate, service credit request, service annual/monthly up time”, etc. [28]. These become an individual complete service DP specification category.

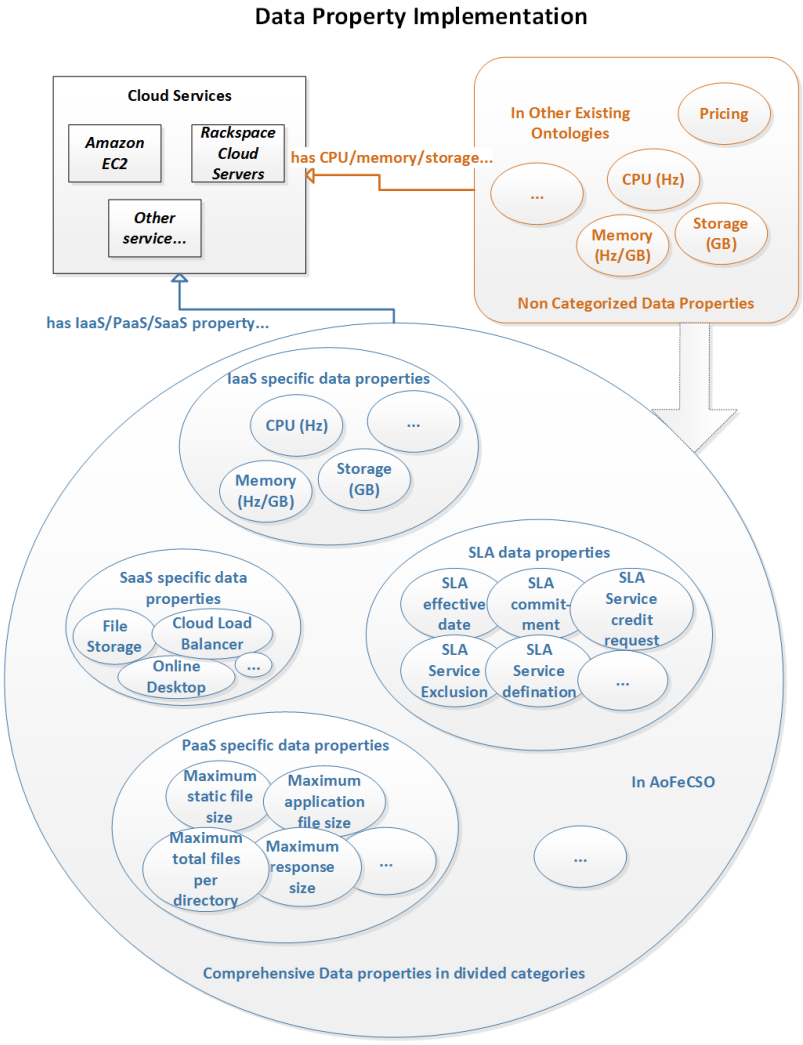


Figure 4.3 Advances of AoFeCSO in Dealing with Data Properties

4.1.3.4 Multi-sourced Assertion of Cloud Entity Annotation Properties

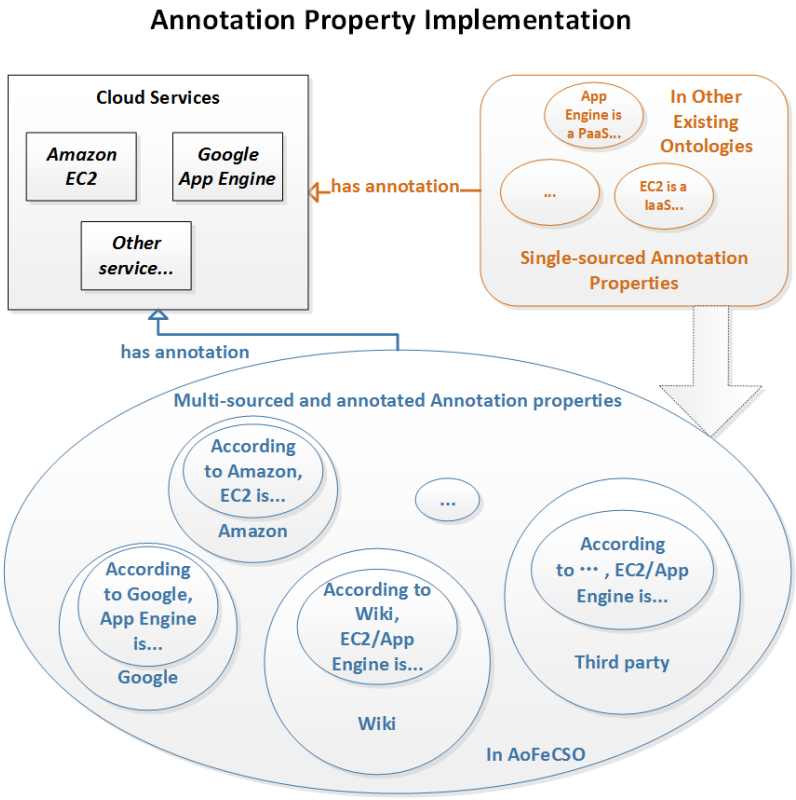


Figure 4.4 Advances of AoFeCSO in Dealing with Annotation Properties

As depicted in Figure 4.4, AoFeCSO utilises annotation properties in a rather different approach against [121, 122, 128] for concept annotations. It involves annotating not only cloud services, but also all other entities appeared in the ontology, e.g. service delivery/deployment models, service characteristics, service properties, cloud service companies, OSs, programming languages, protocols, APIs, etc., regardless of their uniqueness or commonness. In this way, the whole ontology becomes much more interpretable, even to non-expert users.

Moreover, unlike others who acquire (annotation) information from a single knowledge source, AoFeCSO collects and uses multiple, in fact, as many as possible, descriptions over a diversity of sources. This establishes trustful concept annotations throughout the ontology, since each annotation asserted is

accompanied with its origin source information (by annotating the annotation with the source data). Obviously, these multi-sourced annotations offer a much more comprehensive view for the modelled entities, and by interpreting which users would gain more insights than they could from any single one.

4.2 Fuzzy Cloud Service Specification with OWL2 Fuzzy Extension

4.2.1 Fuzzy Scenarios

The fundamental elements of OWL are individuals, classes, OPs and DPs [126]. OWL2 fuzzy extension can therefore be achieved if the fuzziness of the above basic elements and their relations are addressed. More specifically, it deals with the following three scenarios: the fuzzy subsumption exists in individuals/classes/OPs/DPs, the fuzzy restrictions asserted on individuals or classes of individuals, and the fuzzy values used in axioms (literal, secondary individuals or classes of individuals). In the following fuzzy scenarios, "$()^{I}$" denotes an individual, "$()^{C}$" denotes a class, "$()^{OP}$" denotes an OP, "$()^{DP}$" denotes a DP. "$\in$" denotes to fuzzily belong to, "$\subseteq$" denotes to fuzzily subsume.

4.2.1.1 Scenario 1: Subsumption Weights

Based on the fuzzy set and PLN theories talked in Section 2, "subsumption weights" are introduced to illustrate at what degree a class/property/superclass subsume an instance/subpropety/class in OWL2. The weight works on top of formal OWL2 sub/subsume assertion and does not tend to modify the overall ontology hierarchy (for now). For example, if an instance is specified to belong to either set *A* or set *B*, different subsumption weights can express which set the instance is more or less likely to belong to; or if both *x* and *y* belong to the same set, dissimilar subsumption weights can indicate which one follows the maximum specification of the set.

A. Subsumption weights of OWL2 individuals

Case 1:

Individual $(a)^I$ fuzzily belongs to only class $(C_1)^C$ which disjoints its sibling classes $(C_i)^C$ within their superclass $(C)^C$; the subsumption weight is $\mu_{(C1)}{}^c((a)^I) \rightarrow (0, 1]$, which satisfies that the degree of $(a)^I$ belonging to the superclass $(C)^C$ is at least the degree of $(a)^I$ belonging to $(C_1)^C$ and up to *100%*; denotes:

If $(a)^I \in (C_1)^C$ *and* $(a)^I \notin (C_i)^C$ *and* $(C)^C = (C_1)^C \cup (C_2)^C \cup \ldots \cup (C_n)^C$;

then $\mu_{(C1)}{}^c((a)^I) \rightarrow (0, 1]$;

and $\mu_{(C)}{}^c((a)^I) \rightarrow [\mu_{(C1)}{}^c((a)^I), 1]$;

where for each $2 \leq i \leq n$.

Case 2:

Individual $(a)^I$ fuzzily belongs to either class $(C_1)^C$ or $(C_2)^C$ or … or $(C_i)^C$; where $(C_1)^C$, $(C_2)^C$, …, $(C_i)^C$ can be either disjointed or jointed classes within their superclass $(C)^C$; the subsumption weights are $\mu_{(C1)}{}^c((a)^I) \rightarrow (0, 1]$, $\mu_{(C2)}{}^c((a)^I) \rightarrow (0, 1]$, …, $\mu_{(Ci)}{}^c((a)^I) \rightarrow (0, 1]$, which satisfy that the degree of $(a)^I$ belonging to the superclass $(C)^C$ is *100%*, whereas the sum of all the subsumption degrees is *100%*; denotes:

If $(a)^I \in (C_1)^C$ *or* $(a)^I \in (C_2)^C$ *or … or* $(a)^I \in (C_i)^C$ *and* $(a)^I \notin (C_j)^C \cap (C_k)^C$ *and* $(C)^C = (C_1)^C \cup (C_2)^C \cup \ldots \cup (C_n)^C$;

then $\mu_{(C1)}{}^c((a)^I) \rightarrow (0, 1]$; $\mu_{(C2)}{}^c((a)^I) \rightarrow (0, 1]$; …;

$\mu_{(Ci)}{}^c((a)^I) \rightarrow (0, 1]$;

and $\mu_{(C)}{}^c((a)^I) = 1$;

$\sum_{i=1}^{n} \mu_{(Ci)}{}^c((a)^I) = \mu_{(C1)}{}^c((a)^I) + \mu_{(C2)}{}^c((a)^I) + \ldots + \mu_{(Ci)}{}^c((a)^I) = 1$;

where for each $2 \leq i \leq n$ *and for each* $1 \leq j \leq n$ *and for each* $1 \leq k \leq n$ *and such that* $j \neq k$.

Case 3:

Individual $(a)^I$ fuzzily belongs to the union of class $(C_1)^C$, $(C_2)^C$, …, $(C_i)^C$ within their superclass $(C)^C$; the subsumption weights are $\mu_{(C1)}{}^c((a)^I) \rightarrow (0, 1]$, $\mu_{(C2)}{}^c((a)^I) \rightarrow (0, 1]$, …, $\mu_{(Ci)}{}^c((a)^I) \rightarrow (0, 1]$, which satisfy that the degree of $(a)^I$ belonging to the superclass $(C)^C$ is equal to or greater than the maximum degree of all these subsumptions and up to *100%*, whereas the sum of all the subsumption degrees is in $(0, \sum_{i=1}^{n} i^0]$; denotes:

If $(a)^I \in (C_1)^C$ *and* $(a)^I \in (C_2)^C$ *and …* *and* $(a)^I \in (C_i)^C$ *and* $(C)^C = (C_1)^C \cup (C_2)^C \cup \ldots \cup (C_n)^C$;

then $\mu_{(C1)}{}^c((a)^I) \rightarrow (0, 1]$; $\mu_{(C2)}{}^c((a)^I) \rightarrow (0, 1]$; …;

$\mu_{(Ci)}{}^c((a)^I) \rightarrow (0, 1]$;

and $\mu_{(C)}{}^c((a)^I) \rightarrow [max\{\mu_{(C1)}{}^c((a)^I), \mu_{(C2)}{}^c((a)^I), \ldots, \mu_{(Ci)}{}^c((a)^I)\}, 1]$;

$\sum_{i=1}^{n} \mu_{(Ci)}{}^c((a)^I) = \mu_{(C1)}{}^c((a)^I) + \mu_{(C2)}{}^c((a)^I) + \ldots + \mu_{(Ci)}{}^c((a)^I) \rightarrow (0, \sum_{i=1}^{n} i^0]$;

where for each $2 \leq i \leq n$ *and such that* $j \neq k$.

B. Subsumption weights of OWL2 classes

The subsumption cases are nearly the same as for OWL2 individuals, similarly:

Case 1:

If $(C_0)^C \subseteq (C_1)^C$ *and* $(C_0)^C \nsubseteq (C_i)^C$ *and* $(C)^C = (C_0)^C \cup (C_1)^C \cup \ldots \cup (C_n)^C$;

then $\mu_{(C1)}{}^c : (C_0)^C \rightarrow (0, 1]$;

and $\mu_{(C)}{}^c : (C_0)^C \rightarrow [\mu_{(C1)}{}^c : (C_0)^C, 1]$;

where for each $2 \leq i \leq n$.

If class $(C_0)^C$ is only a subclass of $(C_1)^C$ (fuzzily), the subsumption weight of $(C_1)^C$ subsuming $(C_0)^C$ is $\mu_{(C1)}{}^c : (C_0)^C \rightarrow (0, 1]$, and when $(C_1)^C$ is a subclass of $(C)^C$, the subsumption weight of $(C)^C$ subsuming $(C_0)^C$ is equal to or greater than $\mu_{(C1)}{}^c : (C_0)^C$ and up to *100%*.

Case 2:

If $(C_0)^C \subseteq (C_1)^C$ *or* $(C_0)^C \subseteq (C_2)^C$ *or … or* $(C_0)^C \subseteq (C_i)^C$ *and* $(C_0)^C \cap (C_j)^C \cap (C_k)^C = \emptyset$ *and* $(C)^C = (C_0)^C \cup (C_1)^C \cup \ldots \cup (C_n)^C$*;*

then $\mu_{(C1)}{}^c : (C_0)^C \rightarrow (0, 1]$; $\mu_{(C2)}{}^c : (C_0)^C \rightarrow (0, 1]$;

…; $\mu_{(Ci)}{}^c : (C_0)^C \rightarrow (0, 1]$;

and $\mu_{(C)}{}^c : (C_0)^C = 1$;

$\sum_{i=1}^{n} \mu_{(Ci)}{}^c : (C_0)^C = \mu_{(C1)}{}^c (C_0)^C + \mu_{(C2)}{}^c (C_0)^C + \ldots + \mu_{(Ci)}{}^c (C_0)^C = 1$;

where for each $2 \leq i \leq n$ *and for each* $1 \leq j \leq n$ *and for each* $1 \leq k \leq n$ *and such that* $j \neq k$.

If class $(C_0)^C$ is a subclass of $(C_1)^C$ or $(C_2)^C$ or … or $(C_i)^C$ (fuzzily), the subsumption weights of $(C_1)^C$, $(C_2)^C$, …, $(C_i)^C$ subsuming $(C_0)^C$ are: $\mu_{(C1)}{}^c : (C_0)^C \rightarrow (0, 1]$; $\mu_{(C2)}{}^c : (C_0)^C \rightarrow (0, 1]$; …; $\mu_{(Ci)}{}^c : (C_0)^C \rightarrow (0, 1]$; when $(C)^C$ subsumes $(C_1)^C$ and $(C_2)^C$ and … and $(C_i)^C$, the sum of all the subsumption degrees is *100%*, the subsumption weight of $(C)^C$ subsuming $(C_0)^C$ is *100%.*

Case 3:

If $(C_0)^C \subseteq (C_1)^C \cap (C_2)^C \cap \ldots \cap (C_i)$ *and* $(C)^C = (C_0)^C \cup (C_1)^C \cup \ldots \cup (C_n)^C$*;*

then $\mu_{(C1)}{}^c : (C_0)^C \rightarrow (0, 1]$; $\mu_{(C2)}{}^c : (C_0)^C \rightarrow (0, 1]$;

…; $\mu_{(Ci)}{}^c : (C_0)^C \rightarrow (0, 1]$;

and $\mu_{(C)}{}^c : (C_0)^C = [\max\{\mu_{(C1)}{}^c : (C_0)^C, \mu_{(C2)}{}^c : (C_0)^C, \ldots, \mu_{(Ci)}{}^c : (C_0)^C\}, 1]$;

$\sum_{i=1}^{n} \mu_{(Ci)}{}^c : (C_0)^C = \mu_{(C1)}{}^c (C_0)^C + \mu_{(C2)}{}^c (C_0)^C + \ldots + \mu_{(Ci)}{}^c (C_0)^C \rightarrow (0, \sum_{i=1}^{n} i^0]$;

where for each $2 \leq i \leq n$.

If class $(C_0)^C$ is the subclass of $(C_1)^C$ and $(C_2)^C$ and … and $(C_i)^C$, the subsumption weights of $(C_1)^C$, $(C_2)^C$, …, $(C_i)^C$ subsuming $(C_0)^C$ are: $\mu_{(C1)}{}^c : (C_0)^C \rightarrow (0, 1]$; $\mu_{(C2)}{}^c : (C_0)^C \rightarrow (0, 1]$; …; $\mu_{(Ci)}{}^c : (C_0)^C \rightarrow (0, 1]$; when $(C)^C$ subsumes $(C_1)^C$ and $(C_2)^C$ and … and $(C_i)^C$, the sum of all the subsumption degrees is in

(0, $\sum_{i=1}^{n} i^0$]. The subsumption weight of $(C)^C$ subsuming $(C_0)^C$ is at least the maximum degree of all subsumptions and up to *100%*.

C. Subsumption weights of OWL2 object properties

The subsumption degree of OWL2 OPs is handled differently from above cases. They are fairly simple, since we do not tend to say an OP $(OP_0)^{OP}$ is either a subproperty of $(OP_1)^{OP}$ or $(OP_2)^{OP}$, or $(OP_0)^{OP}$ is both the subproperty of $(OP_1)^{OP}$ and $(OP_2)^{OP}$. For the simplest case, it uses $\mu_{(OPj)}{}^{OP} : (OP_k)^{OP} \rightarrow (0, 1]$ *(where $j \neq k$)* to infer the weight of $(OP_j)^{OP}$ subsuming $(OP_k)^{OP}$. In addition, for the case of multiple OPs belonging to their mutual superproperty, the subsumption weights would hardly interfere with each other. Therefore, for the superproperty $(OP)^{OP}$ which consists $(OP_1)^{OP}$, $(OP_2)^{OP}$, …, $(OP_n)^{OP}$; in implies $\mu_{(OP)}{}^{OP} : (OP_1)^{OP} \rightarrow (0, 1]$, $\mu_{(OP)}{}^{OP} : (OP_2)^{OP} \rightarrow (0, 1]$, …, $\mu_{(OP)}{}^{OP} : (OP_i)^{OP} \rightarrow (0, 1]$, where for each $2 \leq i \leq n$.

D. Subsumption weights of OWL2 datatype properties

Similarly to the condition of OPs, DP subsumptions denote:

$\mu_{(DPj)}{}^{DP} : (DP_k)^{DP} \rightarrow (0, 1]$;

$\mu_{(DP)}{}^{DP} : (DP_1)^{DP} \rightarrow (0, 1]$; $\mu_{(DP)}{}^{DP} : (DP_2)^{DP} \rightarrow (0, 1]$; …; $\mu_{(DP)}{}^{DP} : (DP_i)^{DP} \rightarrow (0, 1]$;

where $(DP)^{DP} = \{(DP_1)^{DP}, (DP_2)^{DP}, \ldots, (DP_n)^{DP}\}$ and for each $2 \leq i \leq n$ such that $j \neq k$.

It implies that each DP subsumption weight is assigned individually whilst they have no interference against each other.

E. Subsumption weight Example (Class hierarchy)

Other than initially defined cloud delivery model Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS),

emerged concepts such as Communication-as-a-Service (CaaS), Hardware-as-a Service (HaaS) are being accepted to some extent. Instead of saying “HaaS is subsumed by IaaS” and “CaaS is subsumed by either IaaS or PaaS”, subsumption weights can therefore be added to express “HaaS is ‘likely’ to be subsumed by IaaS” and “CaaS is ‘less likely’ to be subsumed by IaaS than by PaaS”, e.g.:

$\mu_{(IaaS)}{}^{C} : (HaaS)^{C} = 0.9;$

$\mu_{(IaaS)}{}^{C} : (CaaS)^{C} = 0.4; \mu_{(PaaS)}{}^{C} : (CaaS)^{C} = 0.6;$

where $(CaaS)^{C} \subseteq (IaaS)^{C}$ *or* $(CaaS)^{C} \subseteq (PaaS)^{C}$ *and* $(IaaS)^{C} \cap (PaaS)^{C} = \emptyset$.

4.2.1.2 Scenario 2: Restriction Weights

Despite the fact that OWL2 OP based relation axioms seem to be similar as the fuzzy relationship theory talked in Section 2, it suggests a rather different approach here. In OWL, both OPs and DPs are used to compose axioms that certain individuals/classes may satisfy. These axioms are implemented in the form of restriction properties, which can only be used positively or negatively. In the case of fuzziness, it is difficult to compose the most appropriate restrictions as the “powers” of the axioms are missing. e.g. the user interface of Apple products is “kind of” friendly. Additionally, suppose two individuals are asserted with the same fuzzy restriction(s), without the power indications it is unlikely to tell the differences between them. For example, both ZohoCRM and SalesforceCRM can do Customer Relationship Management (CRM) related tasks, including account/contacts management, task/event tracking, etc.; without a proper degree indication they would seem to be exactly the same to users. Hence, restriction weights (notated as float “λ”) are proposed on certain OWL2 restrictions to indicate the powers/constraints of the unit interval *(0, 1]*. Restriction weights can be applied to both OP and DP.

A. Restriction weights of OWL2 object properties

For OPs applied between specific OWL2 individuals:

$$\lambda_{(OP)}^{OP} : (I_1)^I \times (I_2)^I \rightarrow (0, 1]$$

For OPs applied on specific OWL2 individuals with OWL2 classes of individuals:

$$\lambda_{(OP)}^{OP} : (I)^I \times (C)^C \rightarrow (0, 1]$$

For OPs applied between OWL2 classes of individuals:

$$\lambda_{(OP)}^{OP} : (C_1)^C \times (C_2)^C \rightarrow (0, 1]$$

One example use of OP restriction weights is to clarify relationships which are not likely to be *100%* strong between OWL2 individuals. Amazon Web Services as an example, have a series of subservices (EC2, S3, Elastic MapReduce, Elastic Beanstalk, etc.) which work in conjunction with each other. However, in the case that they do not have full interoperability among all of them, restriction weights can then be attributed to the OP $(worksInConjunctionWith)^{OP}$ meaning the interoperability between certain paired subservices do not come in full. e.g. $\lambda_{(worksInConjunctionWith)}^{OP} : (EC2)^I \times (S3)^I \rightarrow 0.95$; $\lambda_{(worksInConjunctionWith)}^{OP} : (ElasticBeanstalk)^I \times (S3)^I \rightarrow 0.75$. In addition, in the case of existential restriction, restriction weights are to be applied separately at a per restriction basis over the same OP.

B. Restriction weights of OWL2 datatype properties

Same as OPs, restriction weights also work on certain OWL2 DP restrictions which are applied on OWL2 individuals or classes of individuals, indicating the powers of the data based restriction axioms, denote:

For specific OWL2 individuals:

$$\lambda_{(DP)}^{DP} : (I)^I \rightarrow (0, 1]$$

For OWL2 classes of individuals:

$\lambda_{(DP)}^{DP} : (C)^{C} \rightarrow (0, 1]$

4.2.1.3 Scenario 3: Axiom Value Weights

In addition to subsumption and restriction weights, axiom value weights (notated as float "δ") are proposed to address the fuzziness of the secondary OWL2 resources used in certain axioms. In contrast to restriction weights, the vagueness emerges from the values used in axioms to describe the subject entities. Assuming there are two axioms applied to an individual using the same OP/DP but with two fuzzy values (individual/class with OP or literal with DP), if different fuzzy weights are added to those values additionally in the axioms, it can then reveal: one value is more/less applicable than the other to some extent. Axiom value weights are, therefore, to clarify the truth degrees of using certain OWL2 resources as axiom values in certain axioms. They are initialised when there are ambiguity/inconsistent/vagueness values asserted as the axiom value, denotes:

For OP axiom values asserted between individuals and classes of individuals:

$\delta_{(OP)}^{OP} : (I_1)^{I} \circ (I_2)^{I} \rightarrow (0, 1];$

$\delta_{(OP)}^{OP} : (C_1)^{C} \circ (C_2)^{C} \rightarrow (0, 1];$

$\delta_{(OP)}^{OP} : (I)^{I} \circ (C)^{C} \rightarrow (0, 1].$

For DP axiom values asserted on individuals and classes of individuals:

$\delta_{(DP)}^{DP} : (I)^{I} \circ (LT)^{LT} \rightarrow (0, 1];$

$\delta_{(OP)}^{OP} : (C)^{C} \circ (LT)^{LT} \rightarrow (0, 1].$

Axiom value weights, for instance, can be used to indicate the fuzzy comparison of two individuals assigned over the same OP on the subject individual:

$\delta_{(SLAServiceCompensation)}^{OP}$ $:(S3)^I \circ (ServiceCredit\ Request)^I=1.00;$ $\delta_{(SLAServiceCompensation)}^{OP}$: $(iCloud)^I \circ (ServiceCreditRequest)^I=0.8;$ $\delta_{(SLAServiceCompensation)}^{OP}$ $:(iCloud)^I \circ (ITunesVoucher)^I = 0.2.$ It means traditional cloud providers, like Amazon "S3", returns users "service credit request" if "SLA Service compensation" is authorised. Yet, Apple "iCloud" provides additional option, seen as "ITunes voucher". So an axiom value weight can be used to emphasise that for SLA service compensations, ITunes voucher is less likely to be recognised compare to the traditional service credit request.

4.2.2 Native OWL2 Support

OWL2 introduces a series of new features and rationale in addition to previous OWL, notably as new property constructs, enhanced DP expressions and syntactic sugar [53]. The update makes OWL2 ontology more comprehensive as well as simpler to construct. As a matter of fact, it can accomplish above scenarios of fuzzy weights assertions by using different forms of OWL2 DPs along with these updates, whereas they may have the potential to deal with more complicated cases.

OWL2 DP is primarily designed for asserting axioms that cloud entities may comply with. Traditionally it is used to define facts or specifications that certain concepts fulfil specific data based attributes. In this thesis it proposes a series of unique and categorised "weight" DP domains to implement fuzzy assertions. They are asserted to certain fuzzy entities where necessary and seen like fuzzy tags, where OWL2 annotations can be used to explain how the fuzziness is tackled. Rather than writing a complete new built-in [17] or using OWL2 annotation oriented technique [19], our approach is considered to be more naturally embedded in OWL2 whilst it is supported by traditional DL reasoners.

4.2.2.1 Applicability of OWL2 Subsumption Weight

The subsumption weights are asserted in the form of OWL2 DPs, on top of basic OWL2 individual/superclass/superproperty assertions. For each individual/class/property the subject might belong to, an exclusive named

```
Class (owl:subjectclass)
   SubclassOf (
      ObjectIntersectionOf (
         Class (owl:superclass)
         DataHasValue (owl:subsumptionweight (Literal:double))
      )
   )
```

Figure 4.5 OWL2 Fuzzy Subsumption Weight for Individuals

datatype restriction with the float between *(0, 1]* is assigned. It is recommended that the name of the DP should reflect the class/superclass/superproperty, such as the weight DP $(AppleWeight)^{DP}$ for use of the class $(Apple)^{C}$ subsuming the instance $(iCloud)^{I}$, as the subsumption degree is meaningful only towards the subject's class/superclass/superproperty.

```
Class (:CaaS)
   SubclassOf (
      ClassUnionOf (
         IntersectionOf (
            Class (:IaaS)
            DatatypeDefination ((:IaaSWeight)
               DatatypeRestriction (
                  xsd: double hasValue 0.6)
            )
         )
         IntersectionOf (
            Class (:PaaS)
            DatatypeDefination ((:PaaSWeight)
               DatatypeRestriction (
                  xsd: double hasValue 0.4)
               )
            )
         )
      );
DisjointClasses (:IaaS :PaaS)
```

Figure 4.6 OWL2 Fuzzy Subsumption Weight Application for Classes

For individual's fuzziness the assertion is done by applying the weight DP directly just like other datatype axioms. For OWL2 class's fuzzy subsumption, the simplest case (Section 4.2.1.1.B.Case 1) of fuzzy weight can be asserted as in Figure 4.5.

Additionally, a more complex assertion is demonstrated by using the subsumption example above (section 4.2.1.1.E). First, to assert the basic axiom that all individuals of $(CaaS)^C$ are either members of $(IaaS)^C$ or $(PaaS)^C$ and cannot be both of them. Then inside the "ClassUnionof", $(IaaSWeight)^{DP} = 0.4$ and $(PaaSWeight)^{DP} = 0.6$ are applied separately along with their own subclass statement, where two "IntersectionOf" are formed, seen as in Figure 4.6.

Finally, annotations can be asserted to the classes as well as the weight DPs.

4.2.2.2 Applicability of OWL2 Restriction Weight

Restriction weights are implemented by using exclusively and respectively named OWL2 datatype restrictions. In OWL2, for axioms which are applied between classes of individuals, existential or universal restrictions must be declared and specified.

In the case of existential restriction, it is recommended that the name of the weight DP presents all names and values involved. i.e. the weight DP $(EC2worksInConjunctionWithS3Weight)^{DP}$ for use of $(EC2)^I \times (S3)^I$ with $(worksInConjunctionWith)^{OP}$; for universal restriction, it is not necessarily to mention the secondary class (yet still recommended). Furthermore, for OPs/DPs which are directly asserted between/on individuals, both the individuals/values along with the properties need to be disclosed in the weight DPs. Indeed, OWL2 constraint weights only make sense under named DPs which comprise adequate information of the target axioms.

For individuals' restriction fuzziness assertion, it is done by asserting the weight DP directly as well. Below is an example of a class with single fuzzy weighted OP restriction:

Multiple weights which are assigned to multiple restrictions which applied to a single entity are applied by continuing adding the weight DP assertions within "<ObjectIntersectionOf> </ObjectIntersectionOf>" (See Figure 4.7).

4.2.2.3 Applicability of OWL2 Axiom Value Weight

Axiom value weights can also be interpreted and presented using OWL2 DPs. The way they are asserted is the same as for restriction weight assertions. Despite the fact that axiom value weights are applied to the axiom values only, to avoid weights overlapping and differentiate with two other types of weight, the names of the weight DPs should indicate the OPs/DPs plus the values involved in the axioms. For instance, $(SLAServiceCompensationItunesVoucherAVWeight)^{DP}$ for use of $(iCloud)^{I} \times (Itunes\ Voucher)^{I}$ with $(SLAServiceCompensation)^{OP}$.

4.2.3 Fuzzy Specification Application

While traditional OWL ontologies cannot effectively handle and express uncertainties, AoFeCSO employs a series of fuzzy-extended axioms on top of conventional ontology specifications. This significantly enhances the accuracy of the ontology specification and expression by presenting the most appropriate facts. Generally, in contrast with an ordinary axiom which clarifies a crisp fact, a fuzzy-extended axiom depicts the fact that "to a certain degree this is

```
Class (owl:subjectclass)
  ObjectIntersectionOf (
    ObjectSomeValuesFrom (
      (owl:objectproperty) (owl:secondaryclass)
    )
    DatatypeDefination ((owl:restrictionweight)
      DatatypeResriction (
        xsd: double hasValue #double)
    )
  )
```

Figure 4.7 OWL2 Fuzzy Restriction Weight Application

considered to be true". The degree of truth, usually a float of interval (0, 1), is viewed as the fuzzy weight of the axiom assertion. With such weighted assertions, AoFeCSO is able to clarify: e.g. a service owns "partial" of certain properties; a service works "closely" with another service; a service is sometimes and not always regarded as what it is being specified.

4.2.3.1 Fuzziness Notation and Representation

To explain how the impreciseness service specifications are implemented in AoFeCSO under PLN [51] theory, some examples are demonstrated here, using Dropbox [36]. As a cloud storage service that allows users to save, upload, download, synchronise, and share personal files and folders from different OS platforms in a variety of geographic locations, it further enables developers to build additional applications based on the storage platform. To this extent, such services can be regarded to have some PaaS characteristics whilst they would offer properties such as reliability. However, the "PaaS" and "reliability" specifications are vague and may differ from one person to another; this implies that there might be different degrees of acceptance. According to PLN, these are basic first-order and higher-order logical relationship which denotes (using example fuzzy values):

IntensionalInheritance Dropbox PaaS < [0.3, 0.9] 0.8, 10>

Evaluation hasReliability Dropbox < [0.3, 0.9] 0.8, 10>

These reveal that Dropbox is considered to own PaaS characteristics/reliability attribute at a degree within interval of 0.3 and 0.9 with confidence ("credibility") of 0.8 after 10 more observations ("lookahead"). In contrast to FL representation (which presents only a single fuzzy degree value), it comprehensively reveals not only an interval as the range of the fuzzy weight, but also the confidence of this fuzziness plus the number of collected evidences.

4.2.3.2 Fuzzy Data Collection

Table 4.1 Fuzzy Weight Rating Authorisation Control

Authority	Beginner	Intermediate	Advanced	Expert
Fuzzy weight update	×	√	√	√
Fuzzy interval update	×	×	√	√
Crisp fuzzy convention	×	×	×	√

As fuzziness is seen a subjective matter, a closely constructed fuzzy ontology would then become relevantly subjective, and becomes unideal eventually. To this extent, the approach takes the initiative to involve a great deal of users to rate their own perception weight for appropriate axioms in AoFeCSO. This also complies with the data collection and evaluation processes against relevant PLN theories. By using an integrated user-friendly fuzzy rating mechanism, users do not necessarily require any explicit knowledge of knowledge engineering to make the contribution. Here, the reputation management framework [115] is adopted for the user expertise classifications. Then, for different user expertise levels, the approach provides fuzzy rating authorisation for AoFeCSO, based on the authorisation control reference illustrated in Table 4.1.In fact, the user expertise profile values obtained from other categorisation model can be altered if necessary for our approach.

Basically, the lower the user's level (expertise in CC) is, the smaller the degree of change one will trigger. As shown in Table 1: 1) "Beginners" users are not permitted to input/change anything specified in AoFeCSO. 2) Users from "Intermediate" level and up are allowed to donate their own fuzzy ratings according to their understanding for certain specifications. If so, accepted fuzzy rating will trigger a series of ontology update actions, where a new fuzzy value will be recalculated based on the historical fuzzy rating data stored plus the level of the donating user, under relevant PLN theory. 3) In addition, the fuzzy interval will be updated only if the user is at level of advanced or above. 4) Finally, only "Expert" level users are permitted to make an initial fuzzy rating for a certain specification axiom, as this means to convert the regular axiom from

crisp to fuzzy for the first time. The algorithm prevents low level users from making critical changes to AoFeCSO whilst it increases the overall credibility of the occurred fuzzy specifications.

4.2.3.3 Fuzzy axiom assertion and annotation

Figure 4.5 shows the comparison between regular and fuzzy assertions (and fuzzy annotation) plus the respected reasoning outcome in Protégé [62] snapshots. Using Amazon S3 [2] in the example, originally, while controversial evidences show its belonging to IaaS, PaaS and SaaS, it makes no difference from the three regular delivery model assertions. However, if converted into the fuzzy version, the adds-up part would then be able to reveal that the “PaaS” delivery model for Amazon S3 is considered to be vague (minority agrees only) with the overall weighted average value of “0.21200001f” (“f” stands for float).

Here, since the fuzzy part of the extended axiom is created in the format of regular OWL2 DP axiom, after the conversion, the weight-combined axiom becomes an axiom that intersects the original object axiom and its fuzzy weight data axiom, also in standard OWL2 syntax. As a result, the fuzziness-embedded ontology supports native OWL2 DL reasoner such as FaCT++ [152] and HermiT [131] (see the reasoned/inferred axiom in Figure 4.8).

Meantime, apart from the fuzzy weight value added onto the original axiom, complete fuzziness data as well as all historical fuzzy rating information are also stored in the annotation field of the fuzzy-extended axiom (see the Annotations in Figure 4.8). With respect to PLN fuzzy data representation, the “Interval” concludes the fuzzy weight interval of the historical rating ranges; the “Credibility” captures the up-to-date credibility of the fuzzy weight ratings; the “Count”, which indicates the current total number of ratings, is known to be same as the “lookahead” value. Additionally, historical detailed rating data for each eligible user expertise level is stored, which comprises the average values and counts for “Intermediate”, “Advanced” and “Expert” users respectively.

Figure 4.8 Fuzzy Conversion, Annotation and Reasoning in AoFeCSO

Let FW represents fuzzy weight, $C_{overall}$ represents the overall credibility, the equations for fuzzy weight and credibility calculation takes the form:

$$FW = \frac{\overline{R_I}*C_I*N_I+\overline{R_A}*C_A*N_A+\overline{R_E}*C_E*N_E}{C_I*N_I+C_A*N_A+C_E*N_E} \quad (1)$$

$$C_{overall} = \frac{C_I*N_I+C_A*N_A+C_E*N_E}{N_I+N_A+N_E} \quad (2)$$

where

$$\overline{R_I} = \frac{\sum_{i=1}^{N_I} R_{Ii}}{N_I}$$

$$\overline{R_A} = \frac{\sum_{i=1}^{N_A} R_{Ai}}{N_A} \quad (3)$$

$$\overline{R_E} = \frac{\sum_{i=1}^{N_E} R_{Ei}}{N_E}$$

Here $\overline{R_I}$, $\overline{R_A}$, and $\overline{R_E}$ represent the average rating values of "Intermediate", "Advanced" and "Expert" users respectively, for each ratings R_{Ai}, R_{Ii} and R_{Ei}; C_I, C_A and C_E represent the credibility values of each respected user levels; N_I, N_A and N_E represent the number of total ratings of the different user levels. From the equations, it can be seen that whenever a new rating is accepted, the fuzzy weight and overall credibility is recalculated whilst a series of detailed data fields are updated.

4.2.3.4 Fuzzy Axiom Management

Figure 4.9 illustrates the processes of the ontology fuzzy modification flow. While a new fuzzy rating is detected, it is first verified against the authorisation control specified in Section 4.2.

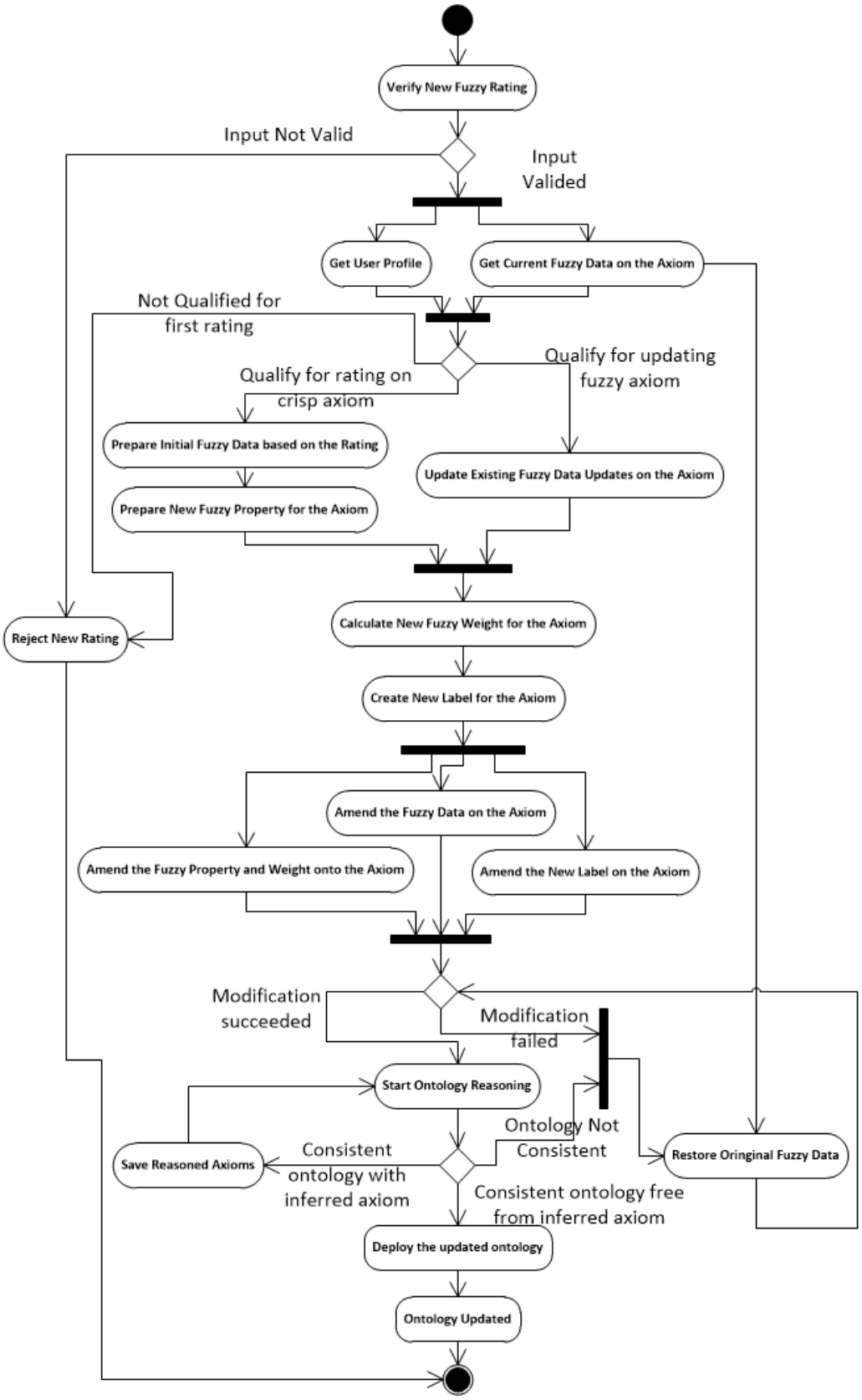


Figure 4.9 Fuzzy Modification Flow

In case of an initial fuzzy weight assertion (crisp-to-fuzzy conversion), a series of fuzziness statements and parameters are created in the format illustrated in Section 4.3 at first. Due to the fact that it is the only rating, the creditability would be 100% whilst the interval is set to +/-10% of the rating value. Followed by that, an ad-hoc DP is created using a name which combines the name of the OP and class plus the word “Weight”, indicating this is a specific restriction applied onto the target axiom. The value of the ad-hoc DP, also known as the fuzzy weight, is simply the rating entered by the expert user.

For fuzzy weight updates, the existing fuzziness data is retrieved and validated at first. Then based on the new rating, appropriate fields are updated according to (3). As the updates complete, a new fuzzy weight and the overall creditability value are recalculated using (1) and (2).

While all fields of the detailed fuzziness data and fuzzy axioms are successfully created/updated, a fuzzy annotation label is also prepared for the fuzzy axiom, based on the new fuzzy parameters as well as the nature of the axiom: e.g. with weight of (0,0.5)/[0.5,1), “STRONG/WEAK” for service property axioms which suggests the cloud service are strongly/weakly considered to own the properties; “DIRECT/INDIRECT” for service functionality axioms, suggesting such is a primary/secondary function of the service; “MAIN/ALSO” for other assertions, meaning that the assertion is mainly/also regarded as such. These further explanations help users better understand the fuzzy weight values with respect to the nature of the information it reveals.

Next, all above updated contents are imported to the ontology where certain data will be changed. If there is no error occurred after updating the ontology, the reasoning process will be initiated to check for any inconsistency or new inferred axioms. Here, any new inferred axioms (new knowledge) will also be saved to the ontology, whereas the original ontology data (if it exists) will be restored if there is any updating/saving error occurred or inconsistency detected.

4.3 Summary

This chapter has presented the design of AoFeCSO. The novel ontology extends the current CC and cloud service semantic modelling practices by embedding two innovative features: agility-oriented design and OWL2 native ontology fuzzy extensions. The agility-oriented design and ontology implementation are able to comprise as many specifications considering the modelling scale and depth. Specifically, this is achieved through the following four tasks: in-depth assertions of cloud service object properties, explicit assertion of cloud service and concept relationships, categorised and compressive assertion of cloud entity data properties, and multi-sourced assertion of cloud entity annotation properties. Further, in order to capture and optimally present the fuzziness appeared in cloud service specifications, AoFeCSO is embedded with a series of fuzzy OWL2 specifications for applicable vague and uncertain descriptions. Accordingly, AoFeCSO can serve as a comprehensive knowledge source to enable effective cloud service search, recommendation, retrieval, plus evaluation and comparison tasks.

Chapter 5 SAMOS

- ***Cloud Service Operation Specification Approach***

This chapter focuses on the semantic presentation of the various cloud service operations. Toward the second object, a unified cloud service access and manipulation operation specification approach (SAMOS) is proposed. The approach achieves generic descriptions for relevant cloud service operation entities, entity classifications, entity relationships, entity data attributes. Incorporating with a series of service operation reasoning mechanisms, SAMOS consequently enables a unified view and manipulation for handling service operation tasks via a single structured interface. Specifically, Section 5.1 describes the three series of service operation specification elements, including specification of cloud service entities and operations, specification of cloud service entity data types, specification of cloud service operational relationships. Then, Section 5.2 demonstrates two service operation verification algorithms that can be used for operation preparation and execution tasks. Subsequently, Section 5.3 introduces a range of service operation reasoning assistances for advanced operation tasks. Finally, Section 5.4 outlines the design of cloud service operation process map for further graphical operation process presentation.

5.1 Modelling Granular Cloud Service Entities and Operations

5.1.1. Specification of Cloud Service Entity and Operation Classification

As a cloud service operation executes, a wide range of entities may be involved. Generally, this can include the subject service, the concepts in the operation condition, the entities involved in the required operation parameters, the objects in the execution outcomes… Obviously, the above cloud service aspects are the fundamental concepts and entities involved in cloud service operations. Among these entities, there are usually certain membership or association

relationships. In fact, these subsumption memberships can be well modelled with ontology classification techniques.

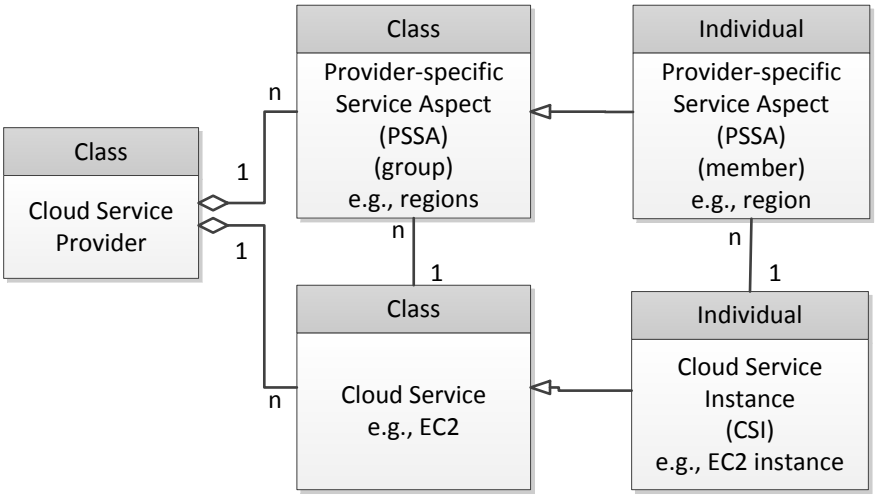


Figure 5.1 Cloud Service Entity Association and Membership Relations

Specifically, as illustrated in Figure 5.1, for cloud services, each cloud service provisioned by a certain service provider can be asserted as an ontology class. According to their subsumption relationships, the service class is modelled as a subclass of the CSP class. Then, any cloud service instance (CSI) created in a cloud service (and owned by a specific service user) is specified as an individual of the service class.

Furthermore, any other concepts and entities involved in a service operation or correlated with a service's attribute are also modelled likewise (See Figure 5.1). This may include possible service operation parameter entities, service configuration specification entities, or a series of service accountability and user authorisation data. Considering that the almost all of such entities tend to be only recognisable for a specific CSP (i.e. the data formats, names, descriptions of the entities would all be unique from one provider to another whilst the entities would only be applicable for certain specific service operation tasks), they are specified as "provider-specific service aspect" (PSSA). Nonetheless, for some common cloud service aspects, despite the distinct PSSA names

being given, they would stand for the same entity fundamentally, e.g. public IP addresses, certain application source files, VM images, SQL database data entries. These associated entities are declared with equivalent class or same individual axioms in SAMOS. Figure 5.2 summarises the association relationships among cloud services, CSIs, PSSAs.

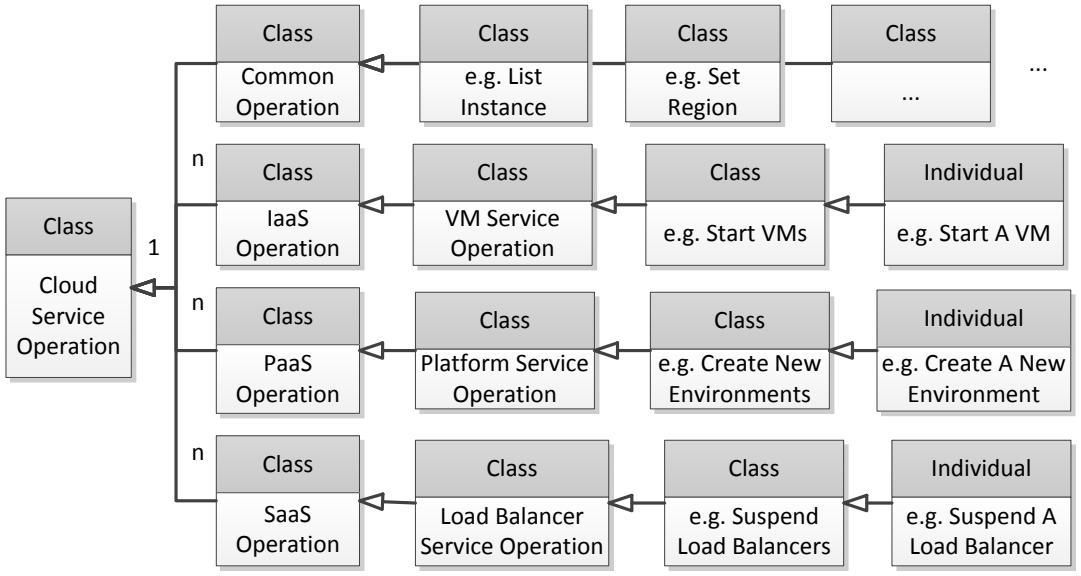


Figure 5.2 Example Cloud Service Operation Membership Relations

In addition to the above, the diverse service operation concepts are also specified for each type of cloud services. As illustrated in Figure 5.2, cloud service operation class comprises four subclasses: IaaS, PaaS, SaaS and common operation. Indeed, due to the distinct service delivery models and functionalities, the majority of the operations would be different from IaaS, PaaS and SaaS. Hence, the three operation classes each have its own list of operations. On the other hand, there are still some general operations which appear to be of no difference among different service models/types. These operations are filled in the common operation class.

5.1.2. Specification of Cloud Service Entity Data Types

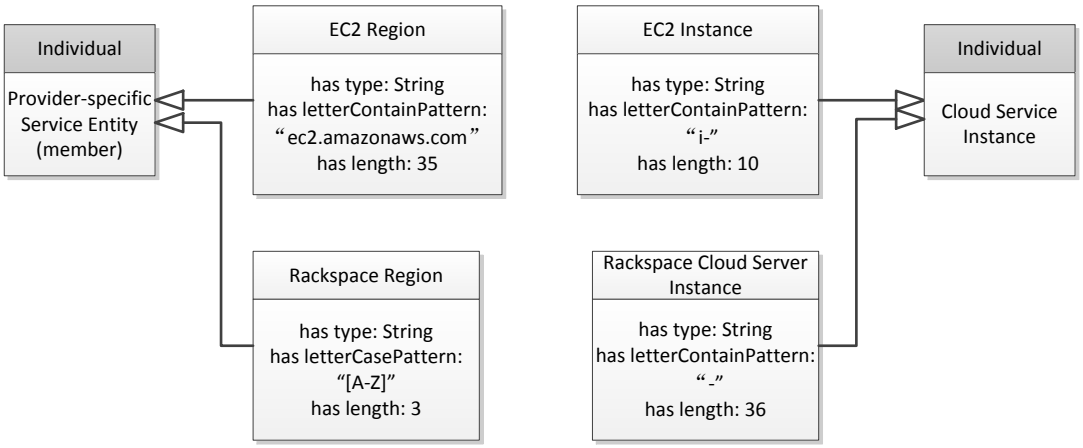


Figure 5.3 Cloud Service Entity Data Type Attribute Implementation

As cloud service concepts and entities are established in appropriate classification hierarchy, their data type attributes are to be specified in details. Generally speaking, cloud services appear to be the same (entity) to everyone, and hence own no typical data-relevant properties. In contrast, the service instances created and owned by users are unique to the owners; they would have certain specific data attributes associated with them. i.e. unique ID, creation time, name, etc. As depicted in Figure 5.3, these details are attached to the instance entities by asserting DP axioms. With the DP assertions, a cloud service instance can be easily recognised by interpreting its unique ID, whereas any other instance attributes can also be effectively addressed as required.

Similarly, all PSSAs are asserted with appropriate DPs according to the respected data formats, e.g. strings, integers, dates, URLs, as well as a series of unique CSP-specific data formats (see Figure 5.3). For instance, for CSPs that offer multiple deployable regions, each set of provider regions would have its own region type information that would make sense only to a specific provider.

Indeed, these DPs enable precise data type format demonstration, differentiation and validation for cloud entities and operations. With the extracted data presentation patterns and respected pattern examination

mechanism, validations can be effectively implemented for cloud service operation preparation and execution (e.g. authorisation, input, output, condition, etc. validations).

5.1.3. Specification of Cloud Service Entity Operational Relationships

Cloud service operations can be seen as reflections of the operational relationships among relevant cloud service and operation entities. For instance, I) “Create instance” and “List instance” can describe the creation and inclusion relationships from a cloud service to its instance(s). II) “Get instance ID” and “Modify instance name” can clarify the retrievable and modifiable relationships from a service instance to its property and condition. This is how SAMOS tackles cloud service operation specification by modelling the diversity of entity operational relationships.

5.1.3.1 Classification of Cloud Service Operations

Shown in Table 5.1, based on the different nature and intentions, cloud service operations are divided into two categories: service information request (SIR) and service manipulation request (SMR).

SIR

At the cloud service level, SIR operations often lie on collecting a cloud service’s available settings and the owned service instances (e.g. get available regions and list instances). At the CSI level, they apply to all kinds of dynamic

Table 5.1 Cloud Service Operation Classification

Service Operation Type	**Description**	**Examples**
Service Information Request (SIR)	Service operation requested to retrieve various service entities and entity information	List owned service instances, get instance ID, get available platforms, etc.
Service Manipulation Request (SMR)	Service operation requested to make changes to cloud services, CSIs or PSSAs	Create new service instance, terminate instance, modify instance name, etc.

instance's running information retrieval (e.g. get instance status). At the PSSA level, they are to acquire the (real-time) information of a specific CSP entity for certain service(s) (e.g. get VM image ID). Generally speaking, these operations usually have a high success rate and take little time for execution.

SIR operations generally require few parameters. The majority of such operations are executed with only a single subject (service, CSI, or PSSA). Considering its main purpose of (dynamic) service information retrieval, the operations typically result into no changes. The returned information can be one or a list of entities, depending on the intension of the request. Overall, SIR operations have few restrictions (e.g. restricted operation frequency, account authorisation control, etc.) for executions.

- *SMR*

At the cloud service level, SMR operations are usually found regarding the general service setting, plus a series of service instance management tasks (e.g. set region and delete all instances). At the CSI level, SMRs make up comprehensive service instance management and configuration function controls (e.g. reboot VM instance). At the PSSA level, it is regarding the manipulations implemented on those unique CSP entities (e.g. delete VM image). On successful execution, SMR should alter the target cloud service/CSI/PSSA in a certain intended way.

Typically, SMR should have certain pre condition requirement. For considerable of such operations, they tend to require some parameters. Obviously, these requirements would vary from different CSPs. As a SMR operation successfully executes, new entities may be produced whilst existing entities may be eliminated, depending on the nature of it. Moreover, the chances of failed SMR operation execution are much higher than that of SIR. Considering the execution time, some SMRs may take a considerably long time; the time may vary dramatically for different time slots.

5.1.3.2 Specification of Cloud Service Operation Object Properties

Based on the proposed operation classifications, the diverse cloud service operations can then be described, shown in the form of various operational relationships among relevant cloud services, CSIs, PSSAs. These relationships can be adequately described using ontology OP assertions. Figure 5.4 illustrates the representation of cloud service operations using OP specifications. Basically, “hasSIR” and “hasSMR” are asserted to describe the types of operations available, between cloud service/CSI/PSSA and relevant operation concepts. For instance, the “create” and “list” operations between cloud services and CSIs can be represented with “hasSMR create instance” and “hasSIR list instance”, respectively; the “get attribute” and “modify” operations between CSIs and PSSAs can be represented with “hasSIR get attribute” and “”hasSMR modify…”, to demonstrate their operational relationships.

Indeed, for almost all types of service operations, there are usually a series of required operation conditions and parameters, whereas successful executions often result into certain outcome(s) (e.g. newly created entities, modified entities, altered conditions, etc.). For different service providers, even for the same operation, these details are likely to be diverse. For instance, for VM instance creation tasks, some IaaS providers may use complex account authorisation restriction and require several VM configuration parameters whilst others may only need few. In order to comprehensively define the various contents involved in the service operations, several systematic operation specification elements are resulted according to their different roles that service the service operations.

As cloud service, service instance, and provider-specific each is modelled with its own list of service operation OP assertions, the detailed systematic operation specifications are stored in the respected OP’s annotation fields. In this way, by interpreting the ontology, it can effectively assist the service request executions for all types of cloud service entities and entity operations.

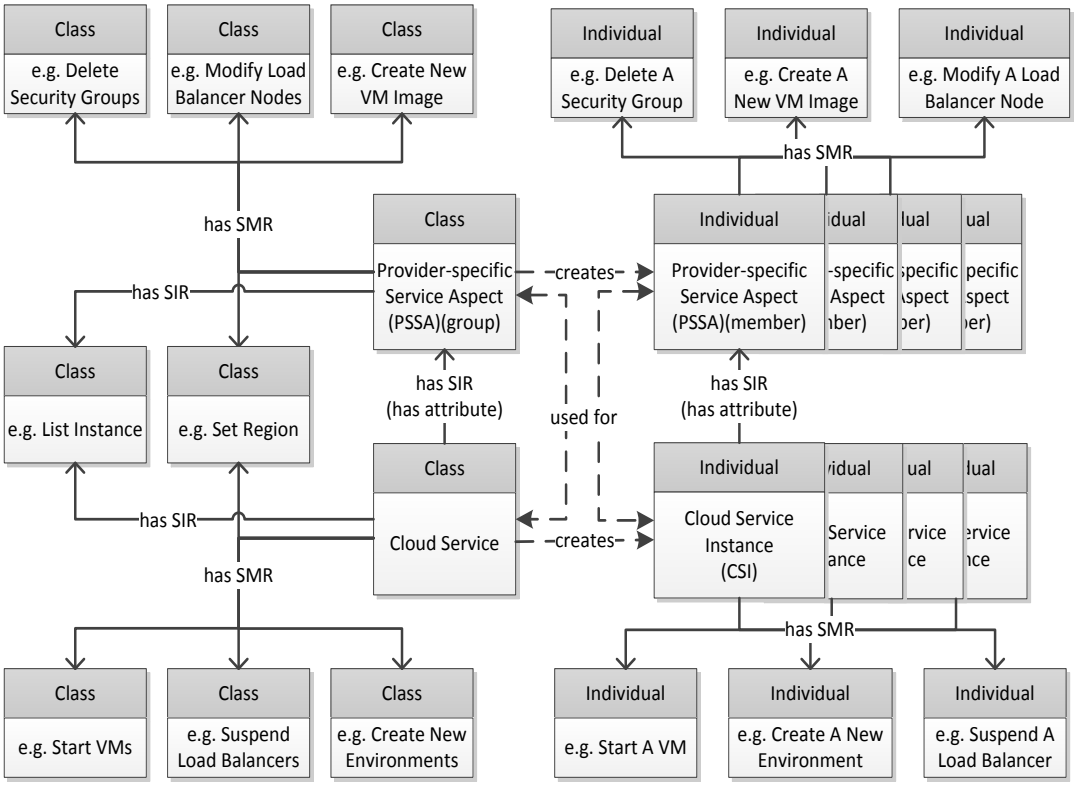


Figure 5.4 Cloud Service Entity Object Properties

5.1.3.3 Specification of Cloud Service Request Operation Elements

Described in Table 5.2, the five cloud service operation elements described are service request subject (SRSubject), service request parameter (SRParameter), service request outcome (SROutcome), service request precondition (SRPreCondition) and service request post-condition (SRPostCondition). With such detailed specifications, any differences among similar service operations would stand out clearly, regardless of across distinct providers or within the same cloud.

- *SRSubject*

SRSubject is recognised as the target that a cloud service operation is implemented over. As a user selects a cloud service for available service operation, the service itself becomes the target. Similarly, as a user chooses a specific CSI for action, the instance is regarded as the target. Additionally,

PSSA entities can also be seen as SRSubject for those operations initiated for them.

Table 5.2 Cloud Service Operation Specification Element

Element Name	Required	Description	Element Format	Element Type
SRSubject	Mandatory	The subject for which a service operation is requested	Single entity	Cloud services, CSIs, PSSAs
SRParameter	Optional	The required parameters that must be satisfied for a service operation request	Single/multiple entities	CSIs, PSSAs
SROutcome	Mandatory	The expected output to be returned after a service operation successfully executes	Single/multiple entities OR "Succeeded/failed"	CSIs, PSSAs OR "null"
SRPre Condition	Mandatory	The condition that must be fulfilled prior to initiating a service operation	Two entities connected with "is" or "isNot" OR "Unconditional"	Cloud services, CSIs, PSSAs OR "null"
SRPost Condition	Mandatory	The expected condition that is formed after a service operation successfully executes	Two entities connected with "is" or "isNot" OR "Unconditional"	Cloud services, CSIs, PSSAs OR "null"

- *Service Request Parameters (SRParameter)*

Although some cloud service operations can be initiated without any information other than the target service/instance, the rest majority do require some parameters, e.g. relevant restrictions, options, customised data, etc. inputted to enable an accurate and successful service operation. SRParameters specifies such details for those applicable service operations. Generally, PSSAs (including the common entities) make up the majority of SRParameters; for some cloud service level operations, CSIs can also be involved as SRParameters (e.g. while attempting to modifying multiple CSIs at once, the selected CSIs are considered to be the parameters of the operation); it is unlikely for any cloud services to be recognised as SRParameters.

To effectively model the various SRParameter requirement specifications, a SRParameter attribute system is developed to specify the series of aspect attributes for each parameter type. This would enable a precise service

operation specification and interpretation while dealing with the variety of the requirement for diverse service operations and parameters. Basically, according to the request requirements of a service operation, a parameter is specified with mandatory/optional differentiations; depending on whether an operation accepts single/multiple parameters of the same entity type, the parameters are also differentiated. The denotations and examples of the SRParameter attributes are shown in Table 5.3.

Table 5.3 SRParameter Symbol Notations

SRParameter Attribute	**Denotation (in OP annotation)**	**Examples Service Operations and SRParameters**
Mandatory	"[]"	
Optional	""	
Single	"()"	
Multiple	"<>"	
Mandatory single	"[()]"	Rename: [(new_name)]
Mandatory multiple	"[<>]"	Reboot: [<vm1,2,…n>]
Optional single	"()"	Set authorisation: (default_security)
Optional multiple	"<>"	Deny access: <user1,2,…n>

- *Service Operation Outcome (SROutcome)*

As an initiated service operation request is handled in the cloud, certain response would be returned from the service provider, informing users whether the operation has been successfully executed, or their expected service information. The operation element is represented as SROutcome. If the purpose of a service operation request is not to acquire/generate any direct service entities, the SROutcome will only be the success result of it; for all other operation requests, SROutcome reveal the expected service entities to be returned from the respected service providers. Typically, a service operation owns only a single SROutcome. Yet, such SROutcome not necessarily has to be one service entity; instead, it can be a series of CSIs or PSSAs as long as they are of the same entity type (represent the same thing in the ontology). For instance, while creating multiple service instances, all new instances become the SROutcome.

- *Service Operation Precondition (SRPreCondition)*

The diversity of cloud service operations lead to various request conditions issues: while some can be initiated without any condition restrictions, others do have specific condition requirements. In order to properly specify the conditions, SRPreCondition defines the mandatory requirement by employing two service entities: in case of a positive condition (e.g. VM is off), the entities are linked with "=="; for a negative condition (e.g. service is not updating), "!=" is used to link the entities; for numbered aspect-involved conditions, ">=" or "<=" is introduced to connects the entities. SRPreCondition specification applies only to those service operations which genuinely require so; others would have an empty entry ("unconditional") for the element.

- *Service Operation Postcondition (SRPostCondition)*

Using the same specification pattern as SRPreCondition, SRPostCondition describes the expected condition changes after a service operation successfully executes. Specifically, this does not necessarily need to contradict with the respected SRPreCondition. In fact, the condition is accounted whenever a "new" condition is formed. For instance, operations of entity creation request would have SRPostConditions such as "Instance is running/active". SRPostCondition specifications can effectively assist requirement preparation for possible subsequent operations.

5.2 Preparation and Invocation of Basic Service Operations

A typical use of the cloud service operation specifications is seen as verification. Given SRSubject, relevant SRParameters and fulfilled SRPreCondition, an operation can be executed through appropriate programming request. If successfully executed, this would result into certain service entity (data) which matches the respected SROutcome and/or SRPostCondition.

5.2.1 Verification of Service Operation Parameters

Specifically, the cloud service operation parameter requirement verification algorithm demonstrated in Figure 5.5 can be used to provide the first control prior to any operation execution. Basically, the service entities, either selected or manually entered data, are processed in two hash lists. One holds the entity type information whilst the other holds the actual data. Here, only the entity type list is used for the verification check: according to the retrieved SRParameter specification, the parameters (types) are verified against the relevant mandatory requirements. As shown in Figure 5.5, whenever there is a parameter type match, the verification counter would increment. Then, if all the mandatory parameters are satisfied, the respected parameter data would be sent to

```
INPUT: Operation op, SRParameterType srpt1, srpt2, ..., srptn;
SRParameterData srpd1, srpd2, ..., srpdn,

 1 INIT SRParameterRequirement srpr1, srpr2, …, srprn to
   CALL getMandatoryParameter with op;
   ParameterSatisfied to FALSE;
 2 IF ParameterCount = 0 THEN
 3     SET ParameterSatisfied to TRUE
 4 END IF
 5 ELSE THEN
 6     INIT matchCount to 0;
 7     FOR each srprn in SRParaterRequirement
 8         IF srprn = srptn THEN /*parameter type matches*/
 9             INCREMENT matchCount
10         END IF
11     END FOR
12     IF matchCount >= ParameterCount THEN
       /*all mandatory parameters satisfied*/
13         SET ParameterSatisfied to TRUE
14         CALL fillParameter with op, srpd1 to srpdn
           /*pass parameter data*/
15     END IF
16 END ELSE

OUTPUT: ParameterSatisfied
```

Figure 5.5 Cloud Service Operation Parameter Verification Algorithm

appropriate operation handler component for further preparation. This then suggests that the parameter verification process is complete.

5.2.2 Verification of Service Operation Preconditions

The service operation precondition verification is implemented depending on the format of the condition specification. As depicted in Figure 5.6, distinct types of SRPreCondition specification are dealt with separately. More specifically, if the candidate requires no SRPreCondition (unconditional), the verification will be complete instantly. Otherwise, a series of dynamically initiated real-time service entity information checks will be implemented. Then, depending on whether a condition is positive, negative or numerical, relevant match or equivalence decision is made against the obtained real-time entity information. Once all mandatory verifications are complete, the operation can then execute as requested.

While the above verifications are mainly for use of simple individual cloud service operations, there are advanced usages, seen as operation assistances. Indeed, the assistances can be widely enabled, such as to automatically prepare preconditions and gather parameters, to program the execution schedules for multiple relevant operations, or to assess the applicability for potential service interactions and compositions. They are provided based on the analysis of cloud service entity operational relationships and the operation specification elements involved.

```
INPUT: Operation op

 1 INIT SRPreconditionRequirement srprec1, srprec2, …,
   Srprecn to CALL getPrecondition with op;
   ConditionSatisfied to FALSE;
 2 IF ConditionCount = 0 THEN
 3     SET ConditionSatisfied to TRUE
 4 END IF
 5 ELSE THEN
 6     INIT SatisfyCount to 0;
 7     FOR each srprecn in SRPreconditionRequirement
 8         INIT condition to CALL
           getCurrentServiceEntityCondition with op, srprecn
 9         IF srprecn HAS "==" THEN
10             IF srprecn = condition THEN /* SRPreCondition
               fulfils certain positive condition requirement */
11                 INCREMENT SatisfyCount
12             END IF
13         END IF
14         ELSE IF srprecn HAS "!=" THEN
15             IF srprecn NOT EQUAL condition THEN /*
               SRPreCondition fulfils certain negative condition
               requirement */
16                 INCREMENT SatisfyCount
17             END IF
18         END ELSE IF
19         ELSE THEN
20             IF  condition COMPLY with srprecn THEN /*
               SRPreCondition fulfils certain numerical (>=/<=)
               condition requirement */
21                 INCREMENT SatisfyCount
22             END IF
23         END ELSE
24     END FOR
25     IF SatisfyCount >= ConditionCount THEN /* all
       preconditions satisfied
26         SET ConditionSatisfied to TRUE
27     END IF
28 END ELSE

OUTPUT: ConditionSatisfied
```

Figure 5.6 Cloud Service Operation Precondition Verification Algorithm

For instance, operation grouping applicability can be analysed by seeking operations with similar types/requirements of both SRParameters and SROutcomes; Operation chaining applicability can be determined when the operations own SROutcome and SRParameter or SRPostCondition and SRPreCondition match (equivalence) one another. The summary of the cloud service operation assistances can be found in Table 5.4.

Table 5.4 Cloud Service Operation Reasoning Assistance Type

Reasoning Assistance Name	**Assistance Description**	**Reasoning Scale**	**Operation Scheduling**	**Precondition & Parameter Preparation**	**Reasoning Steps**
BASR	To assist in preparation of precondition and parameters for unsatisfied service operations	Single cloud (CSP)	None	Guided manual input	1. For the unsatisfied SRParameters and SRPreConditions, list possible options based on current selected SRSubject and SRParameters, plus the real-time status of them;
CCSR	To assist in multiple concurrent service operations of similar types	Multiple clouds (CSPs)	None	Manual input	1. Get SRSubjects' operations which have satisfied SRParameters; 2. Filter the operations based on whether their Preconditions fulfil the real-time SRSubject statuses; 3. Produce the operation lists for the applicable SRSubjects;
SCSR	To assist in automatic scheduled executions of a series of operations in a logical sequence	Single cloud (CSP)	Yes	Manual input	1. Get SRSubjects' operations which have satisfied SRParameters; 2. For the operations, seek for those which have Precondition SRPostCondition matches; 3. Compose these operations into sequenced chains by filtering Them from their factorial combinations, according to the two-two sequenced connections; 4. Filter the operation chains based on whether the first operation's Preconditions fulfil the real-time SRSubject status; Produce the operation lists for the applicable SRSubjects
IOSR	To assist in seeking possible service interactions by linking appropriate operations in a scheduled sequence	Multiple clouds (CSPs)	Yes	Automatic preparation	1. For all SRSubjects' operations, seek for those which have SROutcomes SRParameters (equivalence) matches; 2. For all SRSubjects' operations, seek for those which have SRPreCondition SRPostCondition matches; 3. Compose these operations into sequenced chains as long as their SRPostConditions and SRPreConditions are not contradictory, according to the two-two sequenced connections; 4. Filter the operation chains based on whether the first operation's Preconditions fulfil the real-time SRSubject status; 5. Produce the operation lists for the applicable SRSubjects

5.3 Assisted Service Operation Reasoning

While cloud service entities, their attributes and relationships, and operation elements are comprehensively specified, relevant service operation reasoning can then be introduced to assist cloud service operation tasks. In fact, the reasoning is able to provide dynamic assistances during service operation execution, and even enables cloud service orchestration.

Table 5.4 outlines the four operation reasoning assistances available within SAMOS approach. From their descriptions, it can be seen that they enable better operation experiences and advanced implementation tasks, such as to automatically prepare preconditions and gather parameters, to program the execution schedules of multiple operations, and to assess the applicability for potential service interactions and compositions. Seen in Table 5.4, the assistances are provided based on seeking and analysing the relevant cloud service entities defined in the operation specifications. For instance, operation grouping applicability can be verified via analysing the feasibility of using a single or group of SROutcome as SRParameter for other services' operations, when SRPreCondition complies; interaction applicability can be determined depending on whether the service (instance) subjects have any entities recognised in common.

In the following sections, each of the four assistances is to be discussed in detail.

5.3.1 Basic Assisted Service Request (BASR) Operations

With comprehensive descriptions and detailed specifications of each SIR and SMR for the modelled cloud services, some assistance can be offered dynamically to when users are trying to make the service requests. BASR serves to help users understand and prepare for necessary service request conditions and assist in gathering various request parameters during the request process.

As a user selects some cloud services or service instances, a simple scan of the ontology can firstly bring all available service request options which belonged to them. By looking into the semantically specified SR requirement and outcome data, every SIR/SMR which requires certain request conditions or parameters is accompanied with its own unsatisfied SRPreCondition and SRParameter fields. Then, based on the current selected parameters as well as the real-time service/instance running status, further information regarding how to fulfil the conditions or/and obtain the mandatory parameter can be collected by addressing the relevant entity relationships throughout the ontology. The detailed dynamic request preparation assistance can effectively help users throughout every request process.

BASR is implemented on a per cloud service/service instance and per service request basis. This means that the assistance algorithm does not consider the potential impact resulted from one request to another, or from service instance to another. It is the simplest assisted service request form.

5.3.2 Concurrent Combined Service Request (CCSR) Operations

As previously discussed, for those cloud services and service instances of the same service/instance model/type/function, many of them would share similar service request options, whereas the specification patterns for such requests, i.e. the involved parameters, conditions and outputs often coincide. As a result, given all the necessary request parameters and when the conditions satisfy, a list of service request options can be enabled dynamically for a group of cloud services or service instances for a simultaneously operated request tasks.

CCSR is seen as the dynamically grouped service operations which are to be executed simultaneously for selected cloud services and service instances, when all mandatory request condition and parameters satisfied in prior. The action is made available at a per service request basis. Basically, the reasoning algorithm firstly extracts a list of candidate request options for which the current selected SRParameter and SRSubject are fully satisfied. Afterwards, depending

on whether those requests would require SRPreCondition and the format of the condition specification, a series of reasoning results are returned separately. If the candidate requires no SRPreCondition (unconditional), it is to be returned as a finalised CCSO option, which will be accompanied by a series of dynamically retrieved service/instance information (e.g. origin, reference, SRParameter). In case of experiencing a positive SRPreCondition specification (e.g. the service/instance status == “ready”), the reasoning mechanism would check each of the selected service/instance’s real-time condition, and those which fulfil the condition will be returned along with the request option and other dynamically gathered service information and SRParameter. For a negative SRPreCondition (e.g. the service/instance status != “Updating”), the reasoning process is similar, except that it would return the data as long as the dynamically obtained service/instance condition is not equal to the one specified. From these reasoning outcomes, a list of CCSR options is finally produced.

The above reasoning algorithm enables dynamic transformation of single basic service request into a convenient means to initiate multiple similar service request operations for each fulfilled service request and eligible cloud services/instances. CCSR options provide several combined control for efficient service request tasks, which would dramatically improve the overall service access and request experiences, even of across multiple clouds.

For the reason that CCSR is produced at a per request basis, although it acts to control multiple cloud services/service instances, the algorithm still does not consider the potential impact resulted from one request to another. For all reasoned CCSR options, this means that initiation of certain request could violate others’ operating conditions.

5.3.3 Sequenced Chained Service Request (SCSR) Operations

For some service request operations, the conditions defined in their SRPreCondition may happen to match (or potentially equal to) with others’ SRPostCondition specifications. In other words, to initiate a request with certain

pre-selected SRParameters and SRSubjects, another request has to be executed successfully in prior. The ontology consists of various associated SIR and SMR specification among services and requests. Indeed, this can happen for services both within a single cloud and across multiple clouds. As a number of these service requests are interconnected together with appropriate dependencies one another, a sequenced service request chain (SCSR) can be composed dynamically.

SCSR is formed when the following three conditions are satisfied: I) within all available SIRs and/or SMRs of the selected SRSubjects, there are coherent match between the SRPreConditions and SRPostConditions from one to another; II) all SRParameters required for the SIRs and SMRs are satisfied in prior simultaneously. III) the very initial service request of the chain has SRPreCondition that equals to the real-time retrieved service condition. More specifically, to produce such service request chains for current selected SRSubjects and SRParameters, initially, the reasoning mechanism checks for satisfied service request for the selected SRSubjects and SRParameters. Then, from the list of requests, a list of paired service requests is generated from the ontology. For instance, request R2 requires R1 can be revealed as R1R2. Next, all of the paired service requests are connected whenever the SRPreCondition of a pair's former equals another's latter, as long as a composed request chain does not contain any duplicated service requests. Subsequently, based on the real-time service conditions (retrieved dynamically), the list of chains are filtered where any request chain starts with unsatisfied SRPreCondition is to be removed. Finally, these chains are arranged so that users can find the preferred compositions easily, depending on the initial service request, no of requests, ending request, etc.

When all of the required SRParameters present, SCSR can provide assistance for dynamic cloud service composition tasks by present all possibilities of chained service requests.

5.3.4 Interactive Orchestrated Service Request (IOSR) Operations

Until now, all of the above dynamic service request reasoning assistance process only those requests with fully satisfied SRParameter, which has to be selected manually by users. In fact, there is another route for SRParameter collection, i.e. with a successfully executed service request, the SROutcome entity retrieved in real-time can also be used as potential SRParameter for further service requests. Indeed, this provides another means toward dynamic service request orchestrations. IOSR is designed to provide orchestrated service request that requires minimum information from users. Basically, it attempts to gather all necessary service requests together to prepare the SRPreCondition and SRParameter for the service interaction tasks automatically by reasoning and referring the ontology for comprehensive entity and OP specifications. As a result, the appropriately assembled IOSR option may only need users to choose relevant services or instances as SRSubject and require no additional interventions.

In order to enhance the interaction among cloud services, IOSR reasoning algorithm does not simply consider the exact match between service request's SROutcome and SRParameter or between SRPreCondition and SRPostCondition. Instead, it checks whether there can be indirect "equal" relationships between those entities involved the SROutcome and SRParameter; as long as there are no direct conflicts between the SRPreCondition and SRPostCondition, a link can be created (for certain valid interaction intention). Compared with the above reasoning algorithms, this means that it would examine the operation specifications of all cloud services/CSIs/PSSAs across all CSPs. The orchestration points between the interactive pairs of cloud services can be resulted whenever there are "common" entities involved for the service operations: entities that recognisable regardless of CSPs or service types, e.g. a file with a specific extension, which can be obtained from a storage service and then used by another service; dynamically updated database entries retrieved from a database service being used by another service.

5.4 Service Operation Process Maps

The previous sections has demonstrated that SAMOS approach can reveal the detailed specifications of cloud service operations whilst a series of reasoning assistances can be enabled to assist operation tasks deployment. So far, a possible limitation of the approach is that the semantics–based specifications lack of operation internal execution process information. Indeed, for an operation which requires multiple parameters, SRParameter specification provides no sequence verification information regarding the parameter collection, yet such do exist sometimes (e.g. while performing the actions through the official service web portal). Similarly, some operations do incur inner states (transitions) for the involved CSIs/PSSAs, but such details are not addressed. As a solution, service operation process map modelling (SOPMM) is introduced.

For the service operations which involve “complicated” sub-processes, operation process maps (OPMs) are created to visualise the detailed sub processes incurred while preparing and executing cloud service operations. For CSCs, especially new or potential users, this helps them better understand the explicit details of fairly complicated service operations. Further, while distinguishing similar service operations from different CSPs, this would provide a better contrast in addition to the operation specifications.

SOPMM adopts Petri net-based graphic modelling techniques. Basically, in each OPM, the sub-processes are represented by relevant transitions and places, which are linked with arrows according the actual steps incurred during the operation. Here, the transitions denote the various actions to be performed by a user; the places designate the changes/states to be encountered; the arrows indicate the sequenced relationships between the actions and states. OPMs employ XML formatting for standardised information processing.

Table 5.5 Cloud Service OPM Element Representation

<trans id="ID1412310665" explicit="false"> <posattr x="536.000000" y="197.000000"/> <fillattr colour="White" pattern="" filled="false"/> <lineattr colour="Black" thick="1" type="solid"/> <textattr colour="Black" bold="false"/> <text>enter server name</text> <box w="148.000000" h="28.000000"/> <binding x="7.200000" y="-3.000000"/> </trans>	<place id="ID1412310604"> <posattr x="536.000000" y="127.000000"/> <fillattr colour="White" pattern="" filled="false"/> <lineattr colour="Black" thick="1" type="Solid"/> <textattr colour="Black" bold="false"/> <text>server name entered</text> <ellipse w="158.000000" h="40.000000"/> <posattr x="-440.000000" y="103.000000"/> </place>	<arc id="ID1412316096" orientation="TtoP" order="1"> <posattr x="0.000000" y="0.000000"/> <fillattr colour="White" pattern="" filled="false"/> <lineattr colour="Black" thick="1" type="Solid"/> <textattr colour="Black" bold="false"/> <arrowattr headsize="1.200000" currentcyckle="2"/> <transend idref="ID1412310665"/> <placeend idref="ID1412310604"/> </arc>

As illustrated in Table 5.5, each OPM element (transition, place or arc) is designated with a unique ID. Apart from element names, transition and place elements are created with position and shape data so that they can be displayed properly through user interface. The arcs, which join pairs of transitions and places, only require relevant connection end information, plus the orientations of the links. Figure 5.7 demonstrates an example OPM for Amazon EC2, which is captured from the developed prototype tool.

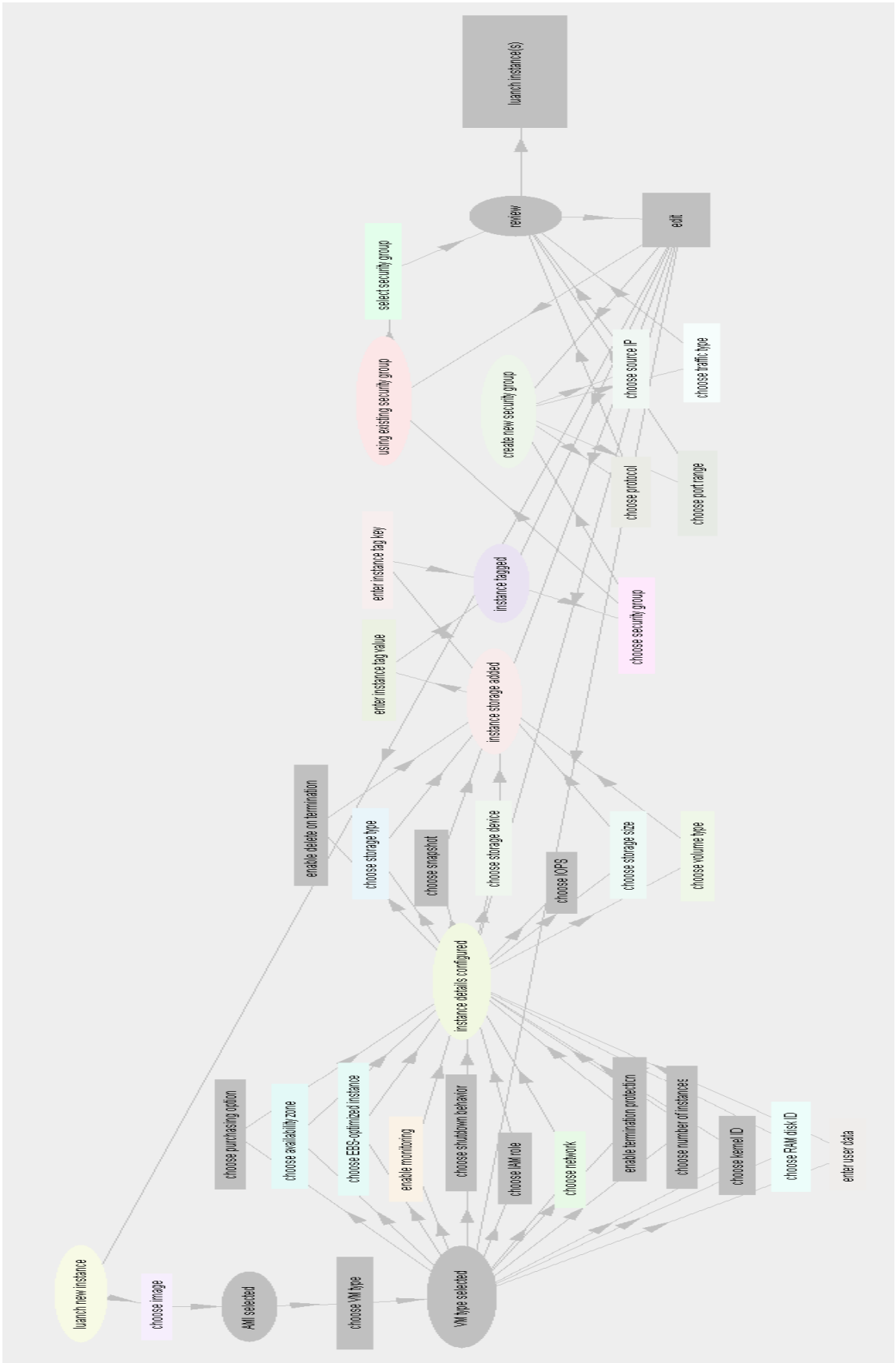


Figure 5.7 Cloud Service OPM Representation

5.5 Summary

This chapter has introduced a new approach for unified clouds service operation specification, known as SAMOS. Resting on ontology modelling techniques, SAMOS is able to describe the diverse cloud service operations comprehensively regardless of the service/operation types. The approach consists of three specification components: specification of cloud service entity and operation classification, specification of cloud service entity data types, specification of cloud service operational relationships. Further, the categorisation in SIR and SMR classifies the wide range of operations according to the operation nature/intent. The details of the involved operation elements, including cloud entity subjects, parameter entities, pre and post conditions, operation outcomes, etc. are specified in granular level so as to present as much as information for every operation. Additionally, benefiting from ontology modelling, SAMOS provides service operation reasoning capabilities for advance use on dynamic operation preparation and validation. The four reasoners, BASR, CCSR, SCSR and IOSR, are able to enable a series of service operation preparation and execution assistances over multiple clouds for a variety of tasks.

Chapter 6 Approach Integration and Process Automation

In Chapter 4 and Chapter 5, the designs of AoFeCSO and SAMOS approach are given respectively. While each of them is developed for presenting a certain range of cloud service specifications, they can be flawlessly integrated so as to combine the diverse CC knowledge as a while for the wide range of cloud service usage tasks. This makes it feasible for providing the versatile usage assistances, including cloud service search, recommendation, retrieval, management and dynamic orchestration, via a united tool interface. As for the third objective, this chapter presents the approach integration and process automation by introducing CSRMP prototype. Firstly, the overall design of the tool architecture is illustrated. Subsequently, the involved sub systems, which each own a separate design and implementation, are described in detail. Finally, the interactions between the sub systems are explained.

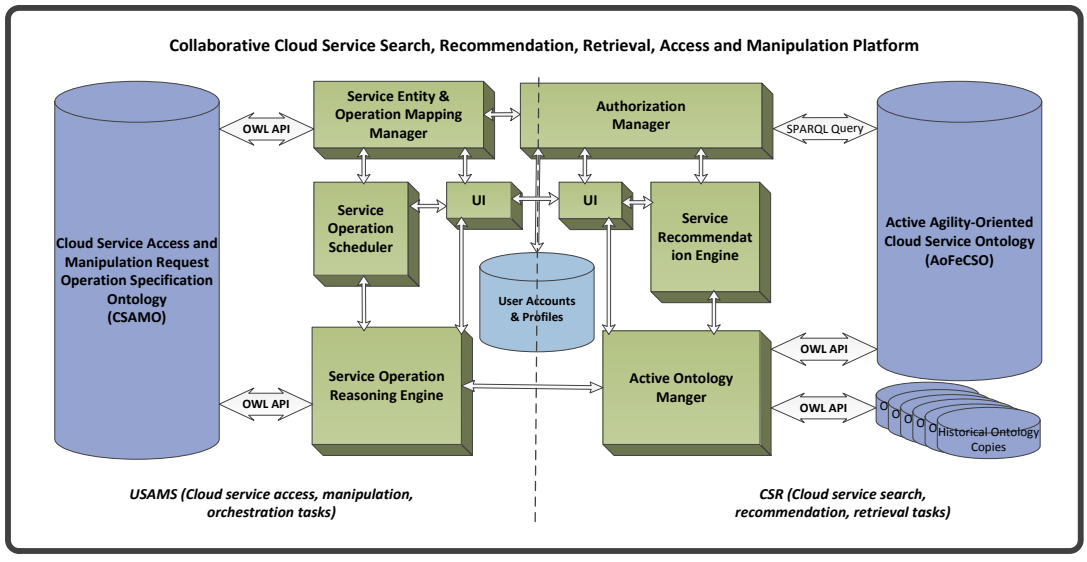
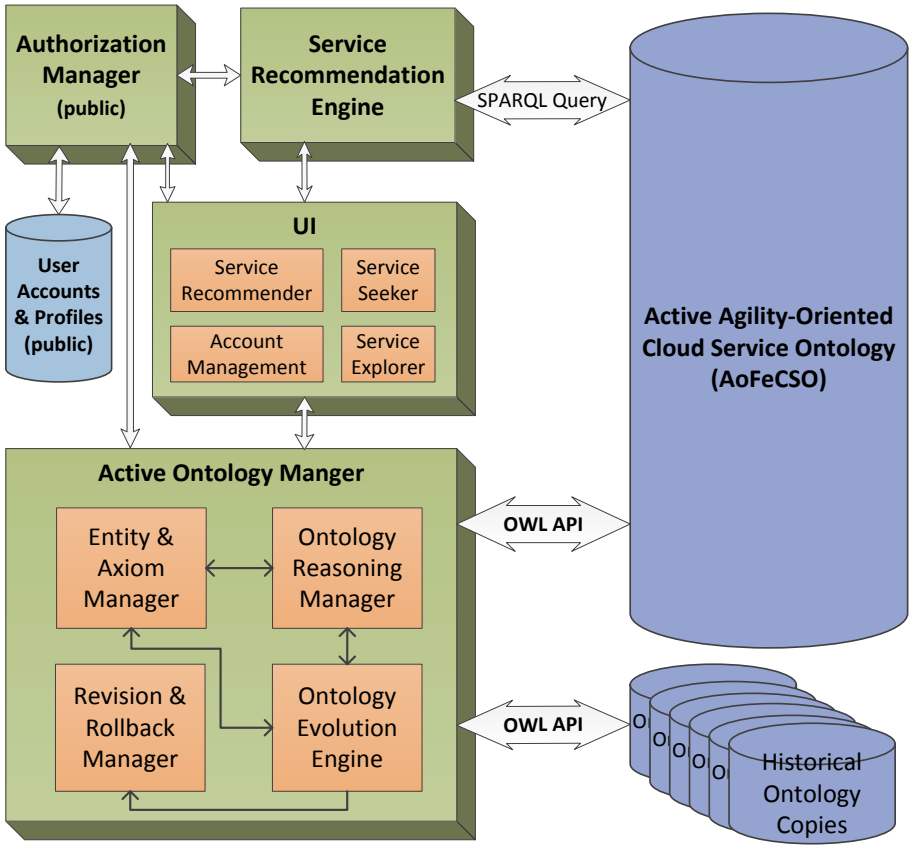


Figure 6.1 CSR Platform Architecture

6.1 Overall Platform Architecture Design

The overall architecture design of CSRMP is illustrated in Figure 6.1. Basically, the platform consists of two sub systems: collaborative cloud service search, recommendation, retrieval system (CSR) and unified cloud service access and manipulation system (USAMS). Each sub system is designed and implemented separately; there are two series of system components which utilise their two ontologies as knowledge sources so as to enable a separate range of cloud service assistance functions. Further, between the two sub systems, there are shared data access and component interactions. The links enables users to navigate from one to another where necessary.

6.2 CSR Sub System Design



CSR (Cloud service search, recommendation, retrieval tasks)

Figure 6.2 CSR Sub System Architecture

CSR comprises Active Ontology Manager, Authorisation Manager, Service Search Recommendation Engine and User Interface four main components (see Figure 6.2).

6.2.1 CSR System Components

- *User Accounts and Profiles Database and Authorisation Manager*

The database stores various user data, which is used as the basis for Authorisation Manager to authorise actions such as service information access, recommendation and fuzzy rating actions. Basically, all registered users can access the ontology specifications through Service Seeker, Service Explorer and Service Recommender. Yet for inputting knowledge, users are given restricted powers against their levels in the authorisation hierarchy.

- *Service Search and Recommendation Engine*

The component takes input of user's preference entries as well as their profiles as the basis to provide service search and recommendation functions. Through pre-set SPARQL query clauses and API queries, the service discovery is implemented by collecting services for exact keywords/filters or both match, whereas the service recommendation is performed by evaluating all services' specifications against a series user specified importance weighted service aspects. When the search/recommendation process is completed, the component outputs a list of services (and additional details) to User Interface.

- *Sub System UI*

UI consists of Account Manager, Service Explorer, Service Recommender and Service Seeker. Account Manager allows users to complete and edit their account and profile details. Service Explorer fulfils various requirements including service information lookup, interpretation, profile analysis, fuzzy rating,

etc. Service Recommender and Service Seeker provide cloud service recommendation and search functions respectively.

- *Active Ontology Manager*

It manages AoFeCSO through OWL API [66]. It incorporates Entity and Axiom Manager, Ontology Reasoning Manager, Ontology Evolution Engine and Revision and Rollback Manager four sub-components. Together they achieve effective ontology management and evolution. Entity and Axiom Manager interprets the ontology axioms whilst can make changes to them by creating a temporary ontology according to certain users' certain requests. It deals with both regular and fuzzy interpretations/modifications two sets of tasks. Ontology Reasoning Manager handles ontology consistency checks and inference controls through binding OWL2 reasoner. The reasoner adopted is FaCT++, due to its faster response plus better syntax and property characteristics support [152]. If the temporary ontology becomes inconsistent after update, the modification will be discarded whilst the details of inconsistency will be reported to the initiator user. This ensures the absolute ontology consistency. Meanwhile, it attempts to discover new knowledge through reasoning inference for ontology evolution purposes. Ontology Evolution Engine is in charge of implementing changes to the ontology: As the reasoning process completes successfully, the consistent temporary ontology plus any new inferred axioms (specification) are saved to replace the current ontology. This is how the ontology evolves progressively while remaining absolute consistency. Revision and Rollback Manager maintains and conserves consistent AoFeCSO redundant copies both actively (called according to schedule) and passively (triggered by certain events). Historical Ontology Copies are guarantees for ontology recovery in case of certain errors occurred while manipulating the current AoFeCSO copy, which could happen during ontology modification, reasoning, saving or replacing processes. If encountered, the most recent copy will be deployed.

6.2.2 Service Search and Filter Rules

The Cloud service search function is implemented based on word/character matches between the query keywords and dynamically retrieved service specifications. On the other hand, the service filter function is provided depending on whether the filter phrases would comply with the obtained service specification clause. More specifically, the algorithm shown in Figure 6.3 describes the rules behind for the functions. Basically, as a use enters a series of keywords to search potential service candidates, the system would scan the specifications from the entire cloud service registry. Whenever there is at least one match, the service is added to the service result list. Further, if a user intends to filter the result (or from the entire services), the system would check if any service that owns specifications which complies with the filters used. Here, only the services that satisfy all the filter phases can be added into the service result list. In fact, the two functions are specifically designed to fulfil certain user needs individually. The search function is for users who want to get as many services as possible, based on some simple information. The filter function is for users who want to get as few services as possible, based on some exact data. Further, the two functions can be used alone or in any preferred orders. In this way, they enable flexible cloud service search functions that would fulfil various needs for a wide range of users with different knowledge background and expertise.

```
INPUT: CloudServiceLitst  csList, KeywordList kwList, FilterList fList
 1. INIT ResultServiceList rsList to null
    //begin search
 2. IF KeywordCount = 0 and rsList = null THEN
 3.    SET rsList  to csList
 4. END IF
 5. ELSE THEN
 6.   FOR each CloudService cs in csList
 7.       INIT CloudServiceSpecifications csSpecs to CALL getSpecifications with cs,
 8.       FOR each keyword in kwList
 9.          IF csSpecs MATCH keyword THEN
10.             ADD cs to rsList
11.          END IF
12.       END FOR
13.    END FOR
14. END ELSE
    //begin filter
15. IF FilterCount > 0 THEN
16.    Set rsList to null
17.    INIT FilterMatch fMatch  to 0
18.    FOR each CloudService cs in rsList
19.       INIT CloudServiceSpecifications csSpecs to CALL getSpecifications with cs,
20.       FOR each filter in fList
21.          IF csSpecs MATCH filter THEN
22.             INCREMENT fMatch
23.          END IF
24.       END FOR
25.       IF fMatch >= fCount THEN
26.          ADD cs to rsList
27.       END IF
28.    END FOR
29. END IF

OUTPUT: ResultServiceList rsList
```

Figure 6.3 Algorithm for Cloud Service Search and Filter

6.2.3 Service Profile (Agility) Evaluation

The evaluation of a cloud service's agility is based on all the specifications that are relevant to the service. An agility score is calculated according to three evaluation criteria. Let PA, SA and TA represent primary tertiary, secondary tertiary and tertiary agility aspects, the assessing equation takes the form:

$$AgilityScore = PA + FW_{SA} * \sum_{I=1}^{N_I} SA_I + \sum_{i=1}^{n_i} TA_i \qquad (1)$$

where N_I and n_i are the total numbers of the secondary and tertiary aspects found, $FW_{SA}$is the asserted fuzzy weight of the aspect.

Basically, primary agility criterion accounts for 50% of a service's agility score, which is determined by the service's function utilities (e.g. resource/platform/software provisions, etc.). Secondary agility criterion takes 40% of the total agility score, which is decided based on the service's main service characteristics and features (e.g. scalability, elasticity, API, OS/programming language support, etc.). Tertiary agility criterion only makes up the rest 10%. It tracks the total number of other service attributes that are regarded weakly relevant to agility (e.g. logging access, application deployment support, migration and transition support, customer service and negotiation support, etc.).

6.2.4 Service Recommendation

Cloud service recommendation is implemented based on user selected weighted recommendation keywords. The process starts by asking for relevant information (keywords) for the target cloud services. The keywords can be of any categories, e.g. services' functions, features, characteristics, etc. The selectable keywords are arranged in a hierarchical layout according to relevant structure/relationships defined in AoFeCSO. Furthermore, to assist users in understanding the unfamiliar terminology, multi-sourced annotation explanations of the keywords are retrieved.

During the selection process, users can specify degrees of importance for each keyword selected. With the list of the weighted recommendation keywords, the recommendation engine scans the ontology and analyses all service specifications for each candidate cloud service. Then for the services which comply with the keywords, recommendation ratios are calculated and provided:

$$Ratio(Service_n) = \frac{\sum_{I=1}^{N_I} I_{K_I} * \sum_{i=1}^{n_i} I_{ki} * FW_i}{n_i} \qquad (2)$$

where I_{K_I} is the main importance degree of the home service keywords category, N_I is the number of the home categories selected, I_{ki} is the sub importance degree of the sub service keywords, FW_i is the fuzzy weight of the encountered service specification if applicable, n_i is the total number of the sub keywords selected for recommendation.

Finally, a recommendation result would contain a list of cloud services. They are accompanied by computed recommendation ratios, indicating how appropriate they would fit the user specified weighted keywords.

6.2.5 Component Interactions

The main interactions among the above system components are seen as: Any ontology modification requests must go through authorisation checks at first. Ontology Reasoning Manager is called every time the ontology is successfully updated, either by Entity and Axiom Manager (due to new information added) or Ontology Evolution Engine (due to new ontology copy saved). Then, 1) if the temporary ontology is inconsistent, it will notify Entity and Axiom Manager to discard the temporary ontology and changes and tell the users the inconsistency along with the cause; 2) if the temporary ontology is consistent and free from new inferred knowledge, it will be forwarded to Ontology Evolution Engine where it will be deployed and take place of the current live ontology; 3) if the temporary ontology is consistent with the updates whilst there are new inferred axioms, the details will be sent back to Entity and Axiom Manager to notify the system user, where upon acceptance the temporary ontology along with the inferred axioms will be saved. Revision and Rollback Manager only receives calls from Ontology Evolution Engine when it fails to deploy the new ontology with the updates. Furthermore, the system components are controlled with a deadlock and queuing mechanism, which prevents possible concurrent

actions during the ontology modification, temporary ontology creation, reasoning processes, and ontology replacement processes.

6.3 USAMS Sub System Architecture Design

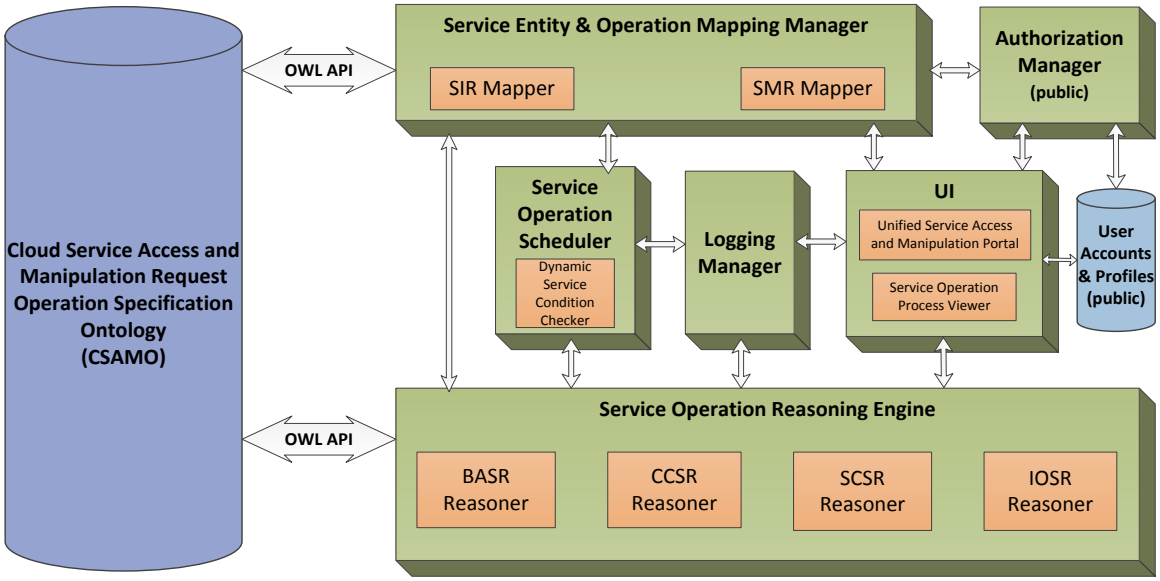


Figure 6.4 USAMS Sub System Architecture

USAMS consists of four main components: User Interface, Service Entity & Operation Mapping Manager, Service Operation Scheduler, Service Operation Reasoning Engine (see Figure 6.4).

6.3.1 USAMS System Components

- *User Accounts and Profiles Database and Authorisation Manager*

In addition to the above data, the database also stores users' cloud service account data (e.g. API credentials), which is mandatory for most of service access and manipulation operations over different clouds.

- *Service Entity & Operation Mapping Manager*

The mapping manager is responsible for retrieving and translating granular service operation and entity specifications and respected API calls and request/response data. This includes interpreting the lists of SIR and SMR operations available for a certain cloud service, plus gathering the entire operation details (i.e. SRPreCondition, SRParameter, SROutcome, etc.) for the operations if required.

Another important function of the component is that it also manages their mapping entries so that users' service operation requests can be implemented properly. Specifically, it has two separate mappers inside for use of SIR and SMR operations respectively. Indeed, due to the many different characteristics between the two operation categories, the operation handling processes are treated separately. This prevents potential issues as attempting to schedule a group of mixed operation tasks.

As a user launches an operation, the component would first obtain the user's API credentials (for the target cloud). Any necessary data format verification tasks are then performed. If no error occurs, the operation task will then be forwarded to Service Operation Scheduler for (scheduled) execution.

- *Sub System UI*

Differently from the above UI which serves for cloud service search, recommendation, retrieval, and evaluation tasks, USAMS UI deals with a wide range of service operation execution tasks by providing a unified portal for real-world cloud service access and manipulation. There are two components involved here: Unified Service Access and Manipulation Portal and Service Operation Process Viewer. Specifically, while the former works to enable a generic portal for various service operation requests and responses, the latter allows users to view the detailed processes incurred for certain complex operations. Primarily using the raw information retrieved from Service Entity & Operation Mapping Manager, it can display the various types of each service operation. Further, for advanced operation tasks such as operation

combinations, the UI provides an interactive means of operation reasoning, dynamic entity information retrieval and operation composition.

- *Service Operation Reasoning Engine*

Service Operation Reasoning Engine incorporates four individual service operation reasoners which each works for a certain operation assistance scenario. BASR Reasoner assists in preparation of the required operation data so as to guide users throughout operation process. The scale of its reasoning is restricted to operations in a single cloud. CCSR Reasoner assists in grouping similar operations for users so as to enable simultaneous executions, even if such are implemented across distinct clouds. SCSR Reasoner assists in scheduling chained tasks when a series of operation are found with certain execution dependency relations one another. Compared to the above, further scheduling control is needed since such operations must be executed (successfully) according to a certain reasoned order. Finally, IOSR Reasoner assists in implementing service orchestration tasks by analysing the possibilities for potential operation interactions for selected services. Its reasoning is implemented with consideration of both dynamically obtained service data and conditions.

- *Service Operation Scheduler*

For advanced service operation tasks, Service Operation Scheduler acts to control the schedule and execution of the involved tasks. The component works closely with Service Operation Reasoning Engine, enabling the tasks reasoned by the reasoners. For every SMR operations executed, the logs are forwarded to Logging Controller.

- *Logging Controller*

Logging Controller documents critical system and operation log entries for users so that they can examine the details at a later time. It fulfils the needs of event

tracking, diagnostics and evaluation for the operations implemented via the platform.

Here, as a user selects any of the returned service information, such would be used as parameters for other service operations. In case of a CSI or certain PSSA returned, it can direct the user to its own list of SIRs and SMR operations

6.3.2 Mapping Ontology Specifications to Service API Calls

Nowadays, cloud services and service resources can be accessed and managed via a diversity of interfaces, e.g. standard web portal, smart phone, tablet cloud service applications, and provider-specific desktop command interfaces. Other than the above, most cloud service providers also release native service API libraries and complete SDKs as a customised service and resource control interface, whereas a series third-party service APIs are also available as an alternative programmable entrance. In fact, these service API call/respond operations can enable more effective service access and enhanced service function manipulation, for the reason that they allow to control services and relevant resources from a much lower level [43]. Since cloud services, service instances, their attributes and relationships are comprehensively modelled, appropriate service API requests can be introduced to the ontology, i.e. mapping service operations specifications with respected API calls, through Service Entity and Operation Mapping Manager (illustrated in Figure 6.5). Here, for a cloud service's OP assertions, each one asserted should be exclusively mapped to a unique API request, where the specified information (e.g. SRParameter, SRPreCondition) must be consistent with the relevant requirements for launching the API request.

As the mapping between the specifications and API calls is established, service operations can be activated and initiated through retrieving interpreting relevant ontology descriptions.

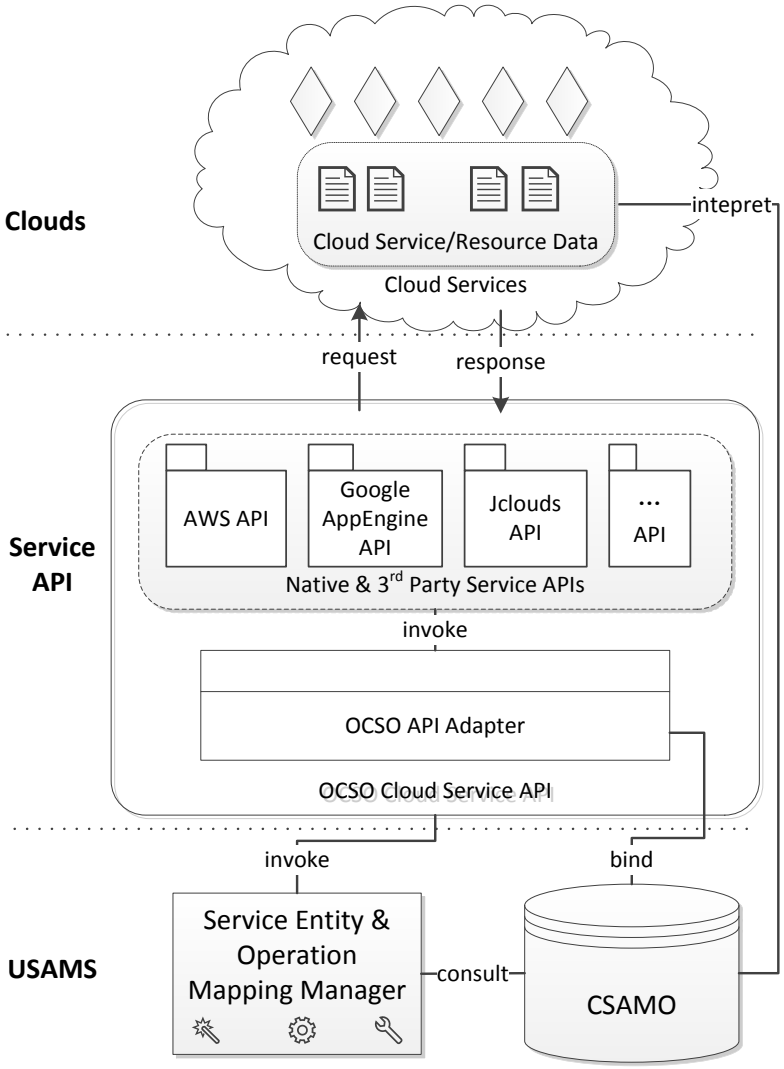


Figure 6.5 The Cloud Service Operation Specification and API Mapping

Given certain cloud services or service instances along with some operation parameters, successfully executed service API requests would result into some dynamically returned service information from the target service's providers. As these responded entities match the defined SROutcome according to the respected OP assertions, depending on their natures, they can either dynamically bring up a new list of available operations (if it is a service or service instance), or be used and reused as SRParameters required for other applicable service's API operations. In this way, by interpreting new entity's OP specifications and possible SRParameter matches, appropriate new service API calls would emerge automatically; once such are executed, further request

options would arise, and so would the future ones. Accordingly, each service entity retrieved from the ontology may result into a dynamically chained manually operated service requests, depending on the intensions of the executed service commands. In fact, due to the dynamic nature of the lively exposed service operation options and the efficient service entity reuse, the mapping greatly enhance the overall service access and control experiences.

6.4 Platform Sub System Interactions

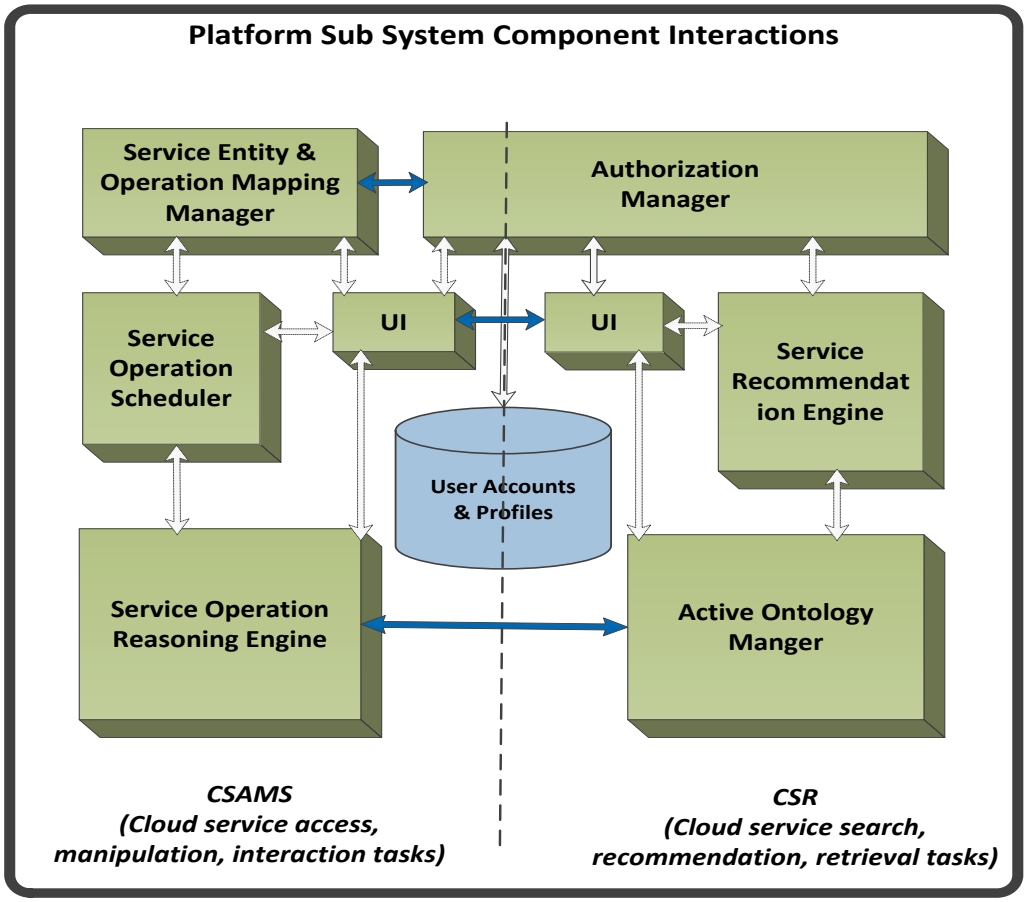


Figure 6.6 CSR Platform Sub System Component Interactions

To enable seamless assistances for a combination of cloud service recommendation and management tasks across the platform sub systems, some of the system components are deployed publically whilst a range of component interactions are enabled (see Figure 6.6). Firstly, the User Accounts & Profiles database and Authorisation Manger are public components that are

shared in use of the two sub systems. Secondly, although there are two separate UIs which specifically work for a set of service assistance tasks, there are diversions available across the UIs so as to allow flexible service views and accesses if needed. Thirdly, another linkage is found between Service Operation Reasoning Engine and Active Ontology Reasoner. This is to enhance the overall assistance function interactions within the platform by incorporating the two ontologies and enlarging the scale of ontology reasoning, e.g. information collected from one ontology can be used for tasks over another.

6.5 Summary

In Chapter 4 and 5, the detailed design of AoFeCSO and SAMOS approach are revealed. Yet, despite their comprehensive knowledge specification and presentation capabilities, it is unclear that whether the approaches can be integrated to provide a combination of cloud service recommendation and management tasks and how these assistances can be offered via the proposed CSRMP prototype. This chapter has given the implementation details with regard to approach integration, prototype tool development and process automation. Towards the aim of a versatile cloud service assistance tool, CSRMP platform has been designed to serve effective and efficient cloud service assistances including service search, recommendation, retrieval, evaluation, access, manipulation, and dynamic orchestration. In the next chapter, a large series of case studies and experiments will be demonstrated in ordered to validate the practical use of the proposed ontologies, approaches and the prototype tool.

Chapter 7 Case Studies

This chapter uses a series of case studies to further demonstrate, validate and evaluate the proposed framework. Specifically, a case study on Google AppEngine is conducted to illustrate how AoFeCSO-based cloud service specifications can be utilised for cloud service search, recommendation and retrieval tasks (via CSR sub system). Then, to demonstrate the practice use and to validate SAMOS approach toward unified service access, manipulation and orchestration, case studies on AWS EC2 and Rackspace Cloud Load Balancers are conducted (via USAMS sub system). Additionally, a number of experiments are implemented to evaluate the performance of CSRMP prototype in terms of the effectiveness and efficiency of the service search, recommendation and retrieval and management functions.

7.1 Agility-oriented Service Search, Retrieval and Recommendation

CSR sub system achieves an effective and effortless means of cloud service search, retrieval, and recommendation through Service Seeker, Service Explorer and Service Recommender interfaces: Service Seeker provides flexible service search and filter options (Figure 7.1); Service Recommender produces service lists and recommendation ratios based on user-defined recommendation conditions (Figure 7.2 and 7.3). Service Explorer divides service specifications into a number of tabs (Figure 7.4, 7.5, 7.6 and 7.7), seen as “General Description, General Attributes, Detailed Attributes and Agility Breakdown”;

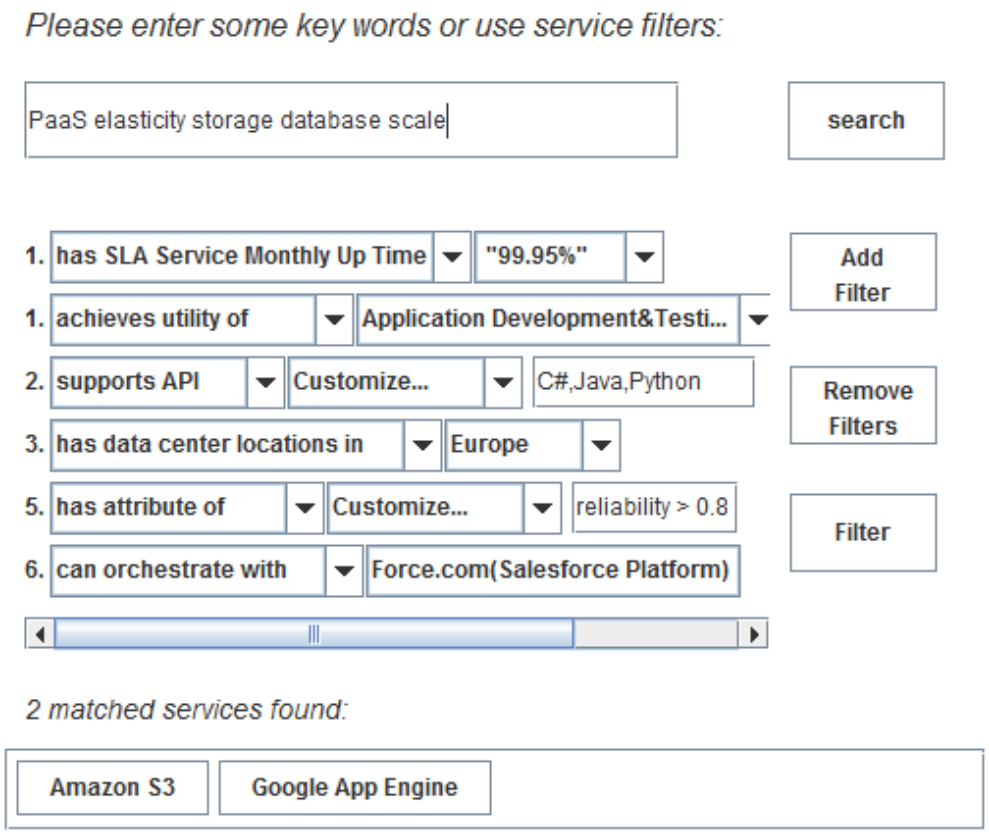


Figure 7.1 Cloud Service Search and Filter

7.1.1 Cloud Service Search with Keywords and Filters

As illustrated in Figure 7.1, CSR Service Seeker accepts both keywords and filters as search options. Users can flexibly use any option alone, or perform one firstly and then apply the other. Taken advantage of AoFeCSO's comprehensive ontological service specification, the keywords can be of a diversity of service concepts, description and attributes and do not necessarily to be explicit. In fact, as long as a service is involved in an assertion which contains the keywords, it is selected as one of the service candidates.

On the other hand, a service filter mechanism is embedded to enhance service discovery. Basically, users are allowed to select certain property restriction clauses so that applicable services can be extracted. The list of properties comprises all available service OP and DP specifications which are retrieved dynamically from AoFeCSO. As an OP/DP is selected, the available property

values are collected and displayed, where users can complete the filter clause by choosing a certain value or entering their customised values. As depicted in Figure 7.1, the customised property value can take diverse forms, e.g. data number ranges, fuzzy assertion weight ranges, strings, etc.

Seen the example search in Figure 7.1, as a user enters “PaaS, elasticity, database, etc.” words, the search would output all cloud services which are specified as PaaS, or with elasticity, or directly/indirectly offers database functions, etc., from applicable CSPs. Then, as a series of filters are deployed, the service lists are reduced based on whether they would fit into each of the restrictions. Users can freely use the given filter terms (which are acquired from AoFeCSO), or insert customise restrictions using numerical values and symbols. As a result, the proposed approach enables much more flexible cloud service search.

7.1.2 Cloud Service Recommendation with Ratios

7.1.2.1 Recommendation Preparation

Figure 7.2 illustrates the appearance of the CSR Service Recommender preparation interface which contains the list of hierarchy displayed keywords, their annotation descriptions and selectable importance factors.

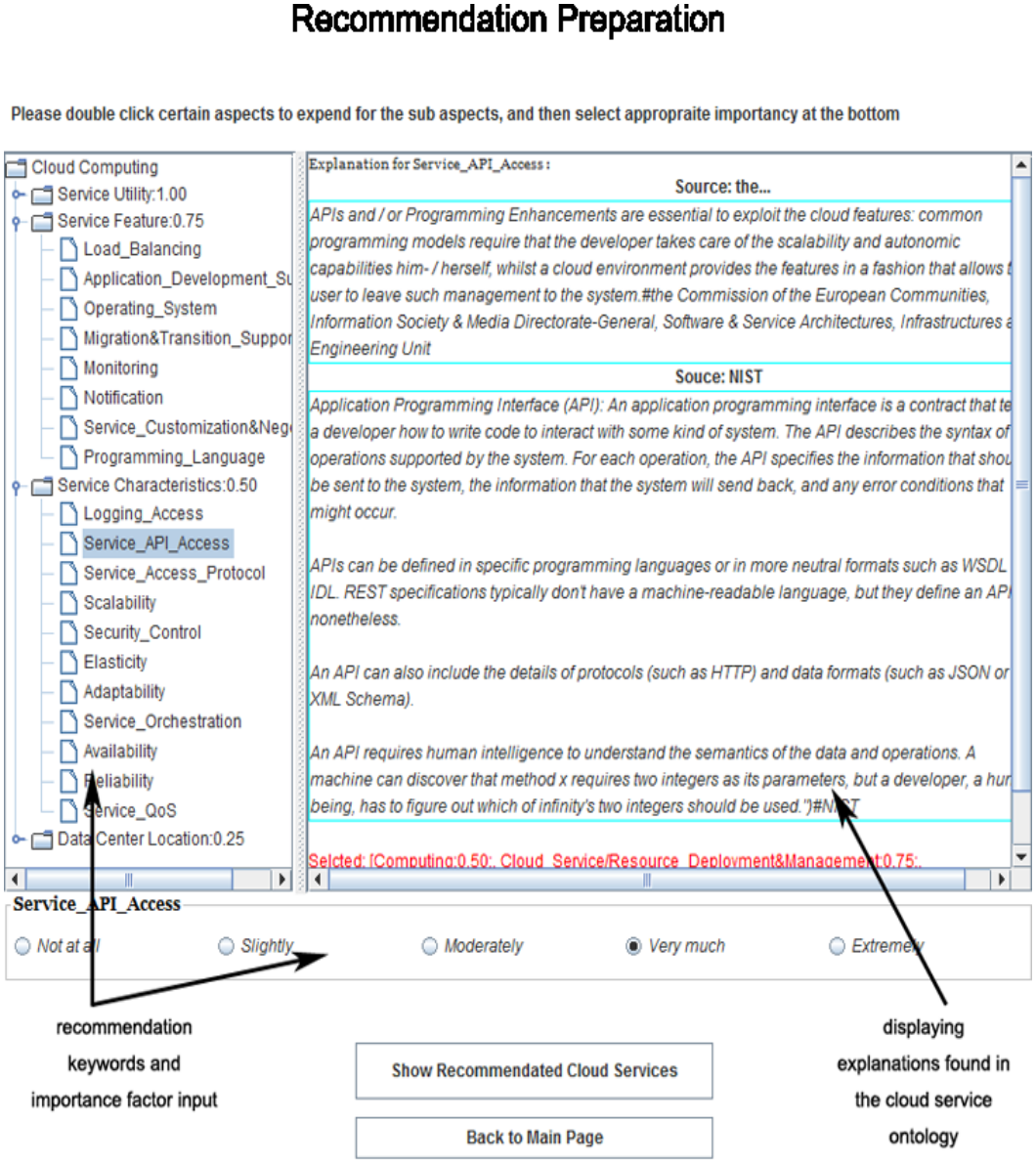


Figure 7.2 Cloud service recommendation preparation

7.1.2.2 Recommendation Result

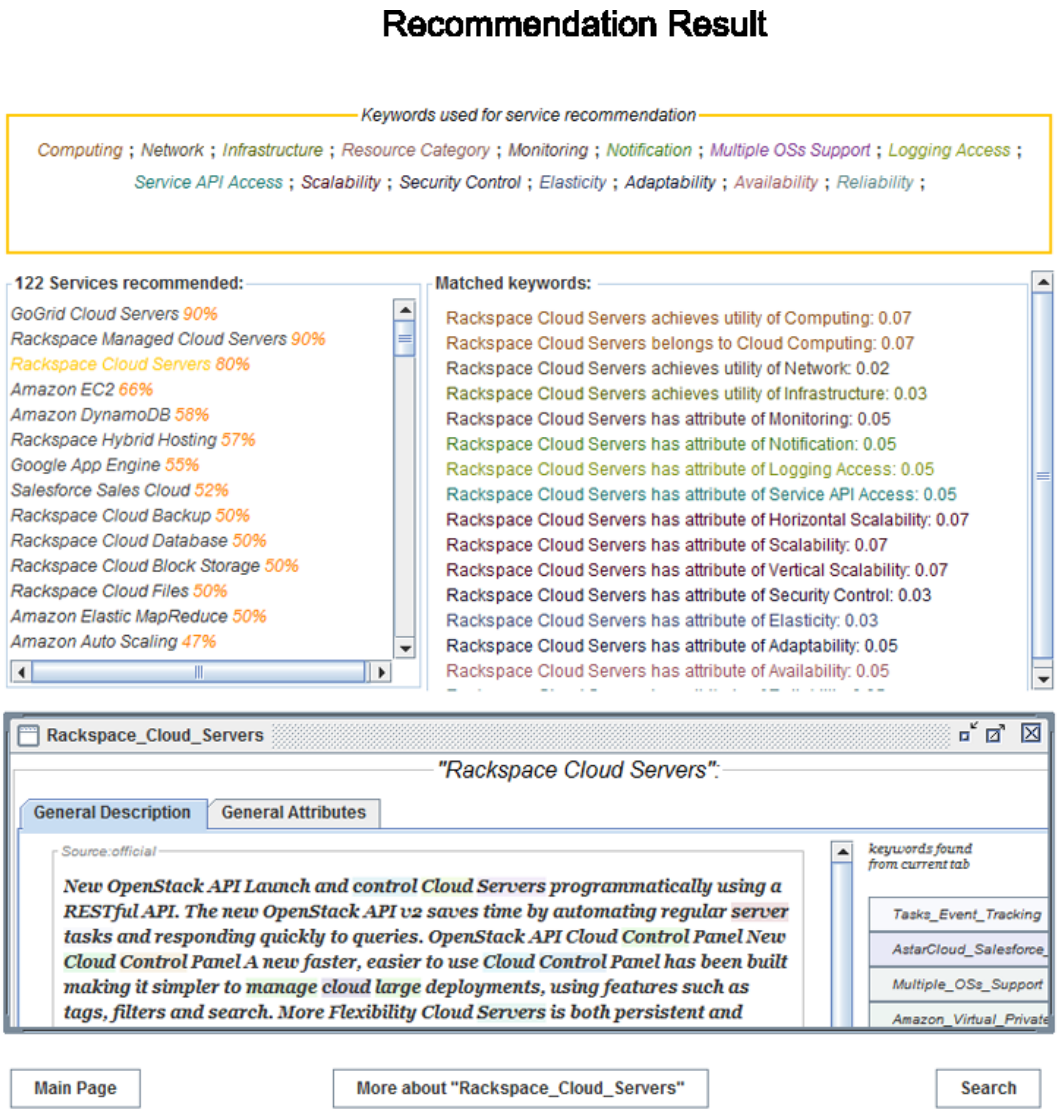


Figure 7.3 Cloud service recommendation result

A recommendation result example is demonstrated in Figure 7.3, in which a list of cloud services is recommended. The keywords used for the recommendation are displayed at the top. The services recommended are accompanied by their recommendation ratios, which indicate how appropriate they would fit the user specified weighted keywords. For more information regarding the recommended service, as the user select a service from the list, the recommendation information for matched keyword and its percentage towards the

recommendation ratio are displayed on its right, whereas a widow containing some service details pops up at the bottom.

7.1.3 Cloud Service Specification Retrieval, Modification and Evaluation

7.1.3.1 General Service Descriptions

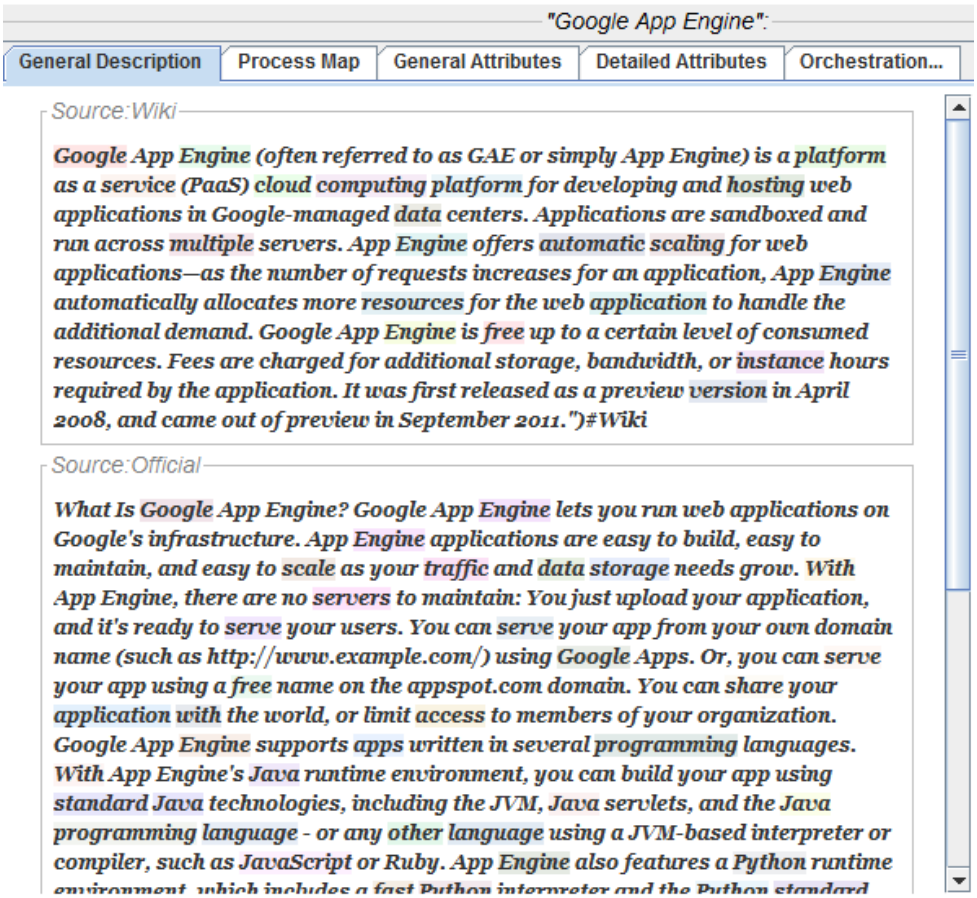


Figure 7.4 Cloud Service General Descriptions

CSR Service Explorer uses “General Description” tab to outline general descriptions of cloud services, which are displayed according to their different origin sources. The service description data is stored in the services’ annotation properties using syntax “rdfs:isDefinedBy” in AoFeCSO, whereas the properties are further annotated with respect source information in syntax “rdfs:comment”.

As demonstrated in Figure 7.4, the retrieved multi-sourced service descriptions can effectively help users understand the services from diverse perspectives.

7.1.3.2 General Service Attributes

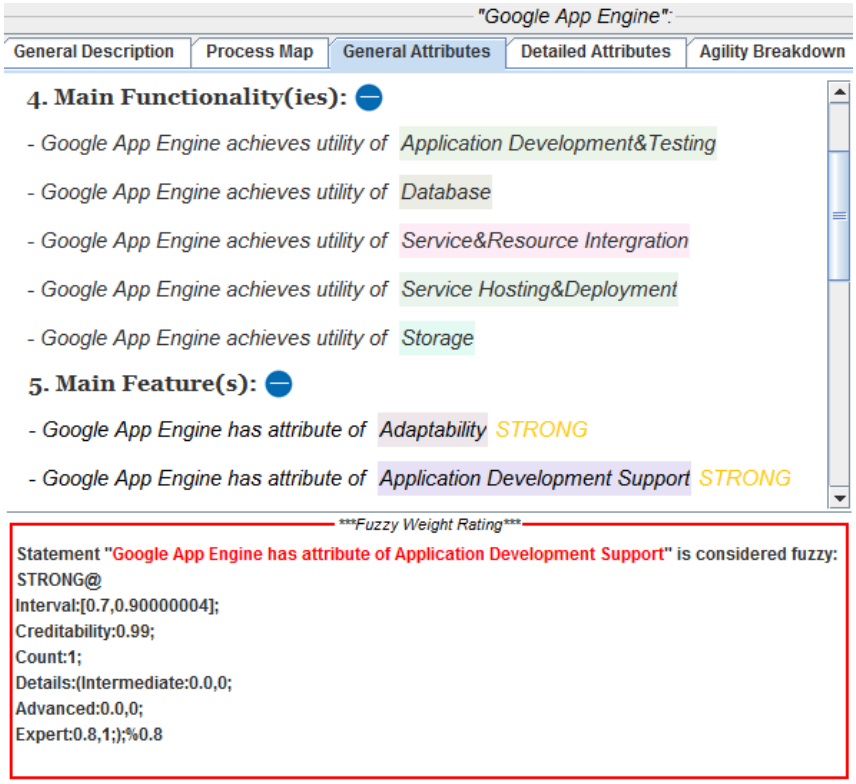


Figure 7.5 Cloud Service General Service Attributes

"General Attributes" tab states a service's general attributes such as its affiliations, delivery model, deployment type, general functionalities and features, etc. (in Figure 7.5). Such information is stored in the form of class assertions (individual-to-class OP assertions) of the service in the ontology. Additionally, for certain information displayed that may not be universally agreed, users are allowed to donate own truth degree ratings based on their understanding or perceptions. Seen in Figure 7.5, the *"MAIN/ALSO", "STRONG/WEAK"* are dynamically created and updated as new fuzzy ratings received, to explicitly reveal the most accurate service attributes towards their rated applicability. The fuzzy modification processes are handled dynamically in the background and do not need any further human intervention, which enables

uninterrupted service search, retrieval and recommendation as well as effortless fuzziness assertion.

7.1.3.3 Detailed Service Attributes

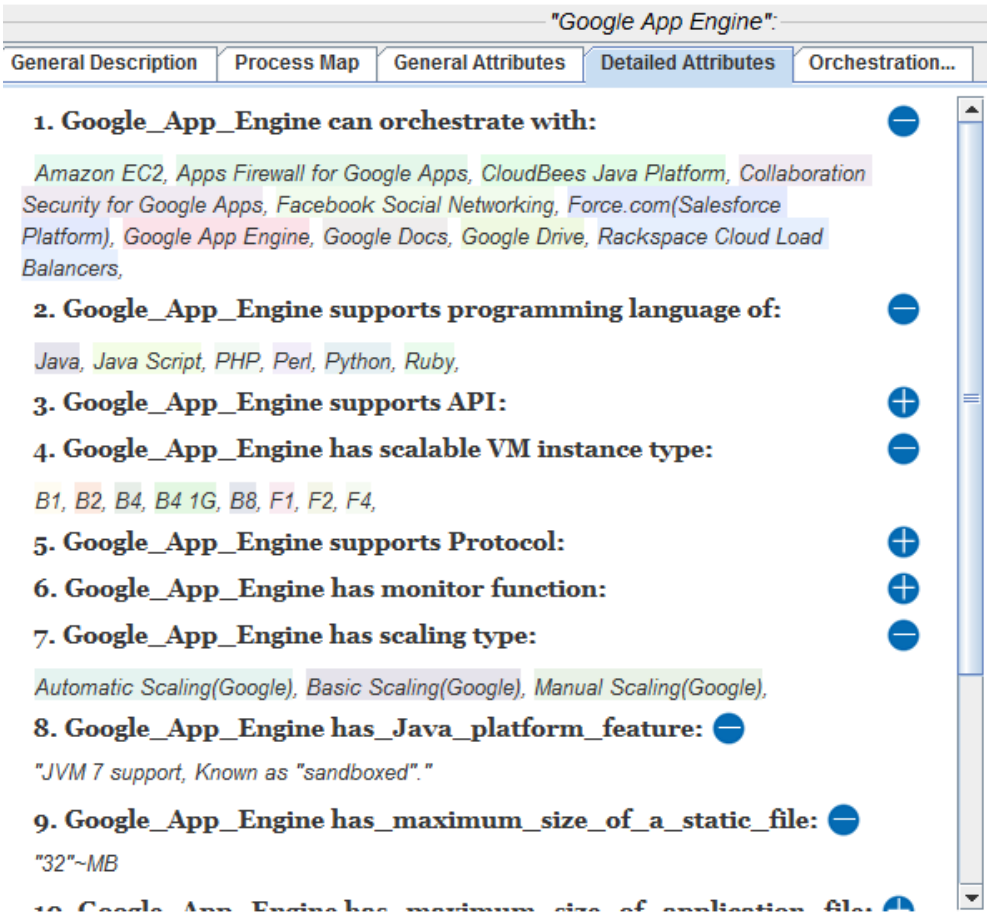


Figure 7.6 Cloud Service Detailed Attributes

“Detailed Attributes” tab in Figure 7.6 outlines the additional details regarding a service’s functions and general service attributes, by translating both the services’ OPs and DPs into explicit statements. In AoFeCSO, such detailed OPs involve a variety of CC concepts and services related individual-to-individual assertions such as: various service orchestrations, supportable OS/API/monitor/security/programming language options for the service, etc. Additionally, the DPs consist of all data-formed service attributes that the service complies such as a service’s operation parameters, constraints, pricing, SLA terms and parameters, etc.

7.1.3.4 Service Profile (agility) Evaluation

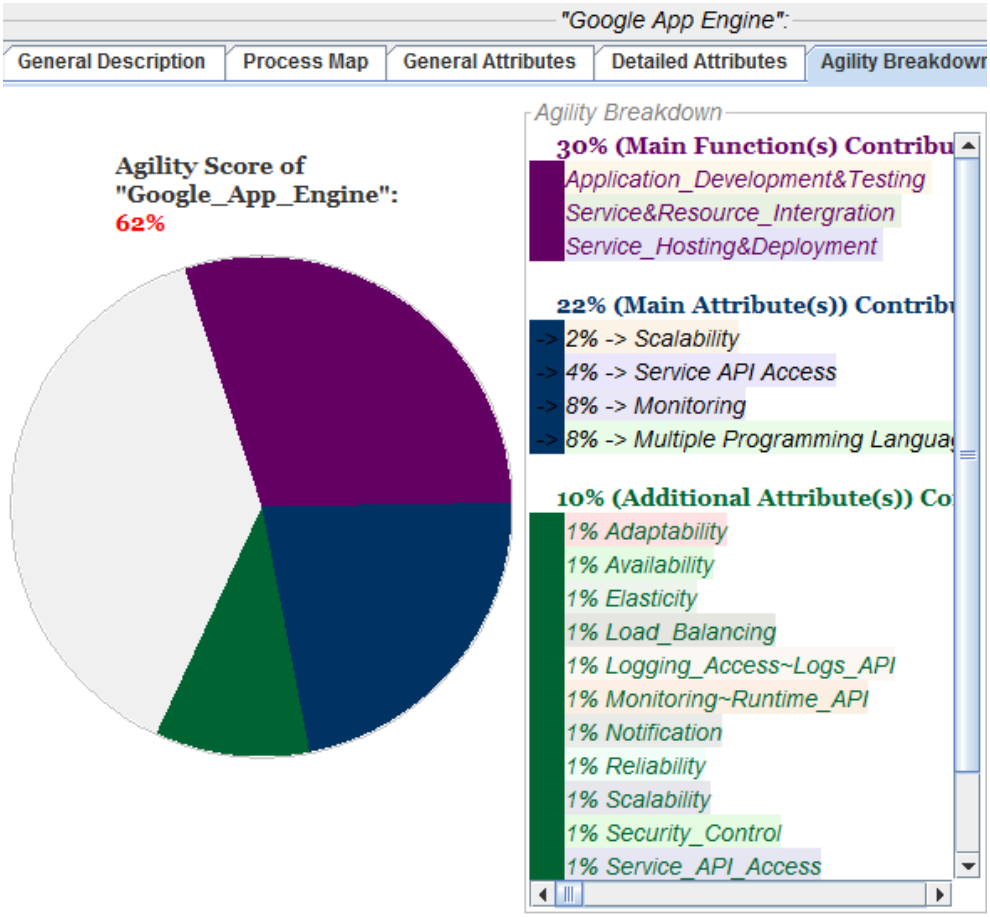


Figure 7.7 Cloud Service Agility Evaluation

In CSR Service Explorer, “Agility Breakdown” tab illustrates an in-depth analysis of a cloud service’s overall service specification. Seen from the example in Figure 7.7, it gathers and analyses all axioms which are relevant to the service, and then generates an agility score plus a series of details which together indicate the service’s overall capability. Here, the main purpose is to demonstrate a method for cloud service agility evaluation. This is to be considered as sample calculation of agility scores, and should be recognised as some guide data. Based on different knowledge grounding and understanding, users may change the agility contributions where necessary.

Primary agility criterion accounts for 50% of a service’s agility score. It is determined by the service’s function utilities. Generally, the rules are: I) the

more resources a service has control over or connection with, the more agile it should be argued; II) the more functions a service can provide, the more agile it ought to be seen; III) the more functional specific a service is designed for, the less agile it should be considered; However, there could be exceptions for certain services, and such are treated differently.

Secondary agility criterion examines whether a cloud service possesses any strongly agility-relevant service characteristics and features. It takes 40% of the total agility score. Overall, those agility-relevant service attributes are seen as: scalability in terms of the scaling types and options, number of available service APIs, options for secure service access and control, number of OSs/programming languages/platform supported, and monitor options. For services that offer better supports or more available options of these service attributes, they would receive higher scores in this against this criterion.

Apart from the attributes presented in primary and secondary agility criterion, there are a number of other service attributes that are regarded as only weakly relevant to agility, e.g. “logging access, application deployment support, migration and transition support, customer service and negotiation support”, etc. The presences of these aspects are assessed for tertiary agility evaluation. This partition only takes 10% of the agility calculation, since such service properties generally have minimal effects towards a service’s agility.

7.1.3.5 Dynamic Keyword Field

Moreover, Service Explorer includes a dynamic list of keywords placed aside for every service information tab according to the contents displayed (the right of the panels in Figure 7.8). The list comprises those CC concepts appear in the tab and are traceable for further information in AoFeCSO. This enables to extract as much knowledge as possible from the ontology in an infinite loop as long as there are further connections among the concepts, and hence significantly increases the quality, quantity and density of service retrieval.

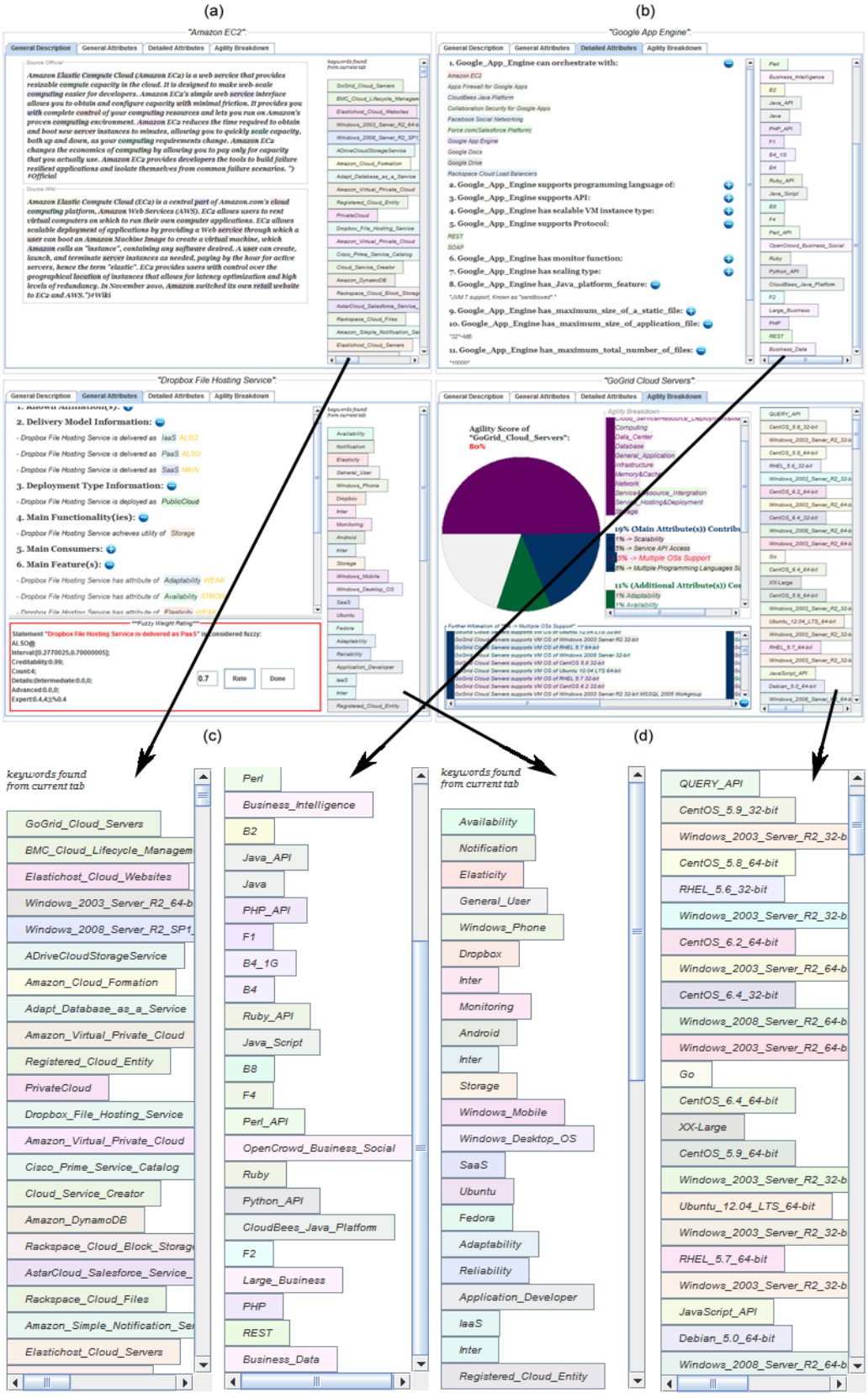


Figure 7.8 Dynamic Keyword Field

7.1.4 Evaluation: Cloud Service Search, Recommendation and Retrieval

Currently, AoFeCSO holds 200 cloud services from more than 100 companies. There are over 20,000 axioms stored in the ontology describing various relationships among cloud services as well as between other CC entities (e.g. service property, granular service entities, fuzzy weights, etc.). To evaluate the proposed approach, a series of experiments are implemented to test both the performance and effectiveness of AoFeCSO and CSR prototype. For ontology evaluation, it discusses a series of aspects according to state-of-the-art ontology evaluation approaches [126, 120].

7.1.4.1 Performance Evaluation

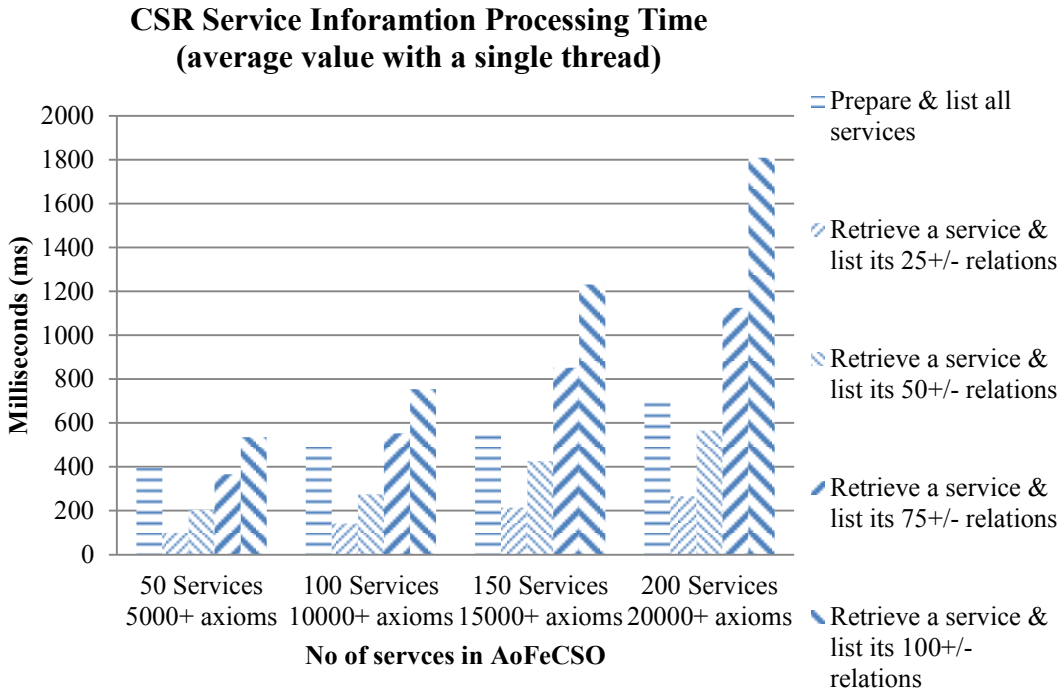


Figure 7.9 CSR Service Information Processing Time

Figure 7.9 demonstrates the performance of CSR prototype as accessing and retrieving the stored cloud service specification from AoFeCSO, when dealing with different total numbers of services (and their data). Reasonably, the larger

the total number of cloud services is, the more time it would require to process their information. More specifically, for preparing and listing all services (and cloud companies), the amounts of time increases are seen fairly gradual: as the total number of services rises from 50 to 200, the time increase is only about 75%. Then, depending on the total relations a service is actively/passively involved in, the individual service access and retrieval time varies rather differently. Firstly, for a service that has around 25 relations (asserted axioms), the time (consumed while accessing it) increases approximately 150% (as from 50 services to 200 services). Secondly, while accessing a service with roughly 50 relations, the time increase is seen as 175%. Thirdly, when there are around 75 relations found for a service, retrieving all the information will need some 210% more time, as the total number of services increases from 50 to 200. Finally, if a service is involved in some 100 relations, the time needed for retrieving all the information would increase approximately 240%, if the total number of services was quadrupled from 50. These statistics suggest that, there is a linear increase for the service access time depending on the total number of services and their service information stored in the ontology, whereas the more relations a service is involved in, the more time it will take while retrieving it with these relations. Fundamentally, this is due to the current single thread processing algorithm used in CSR prototype.

7.1.4.2 Domain Coverage

Table 7.1 Domain Coverage Scale

Coverage	Partial	Full
Infrastructure	Unified business and cloud service ontology [144], Cloud Ontology [73, 59, 169]	FCFA [93], CoCoOn [161], mOSAIC [7], AoFeCSO
Platform	Unified business and cloud service ontology 144, Cloud Ontology 73, 59	mOSAIC [150], Business ontology [90], AoFeCSO
Software	Cloud Ontology [73, 59], Business ontology [99]	[122], Unified business and cloud service ontology [144], cloud software ontology [121], mOSAIC [150], AoFeCSO

In ontology evaluation, domain coverage attempts to justify the ontology knowledge coverage in contrast with other modelling practices (e.g. other gold standard ontologies, information sources, etc.) [119]. Hence, it compares AoFeCSO with a number of existing cloud (service) ontologies in terms of both the scale and details of modelling.

The domain coverage scales of existing cloud (service) ontologies are summarised in Table 7.1. Indeed, the majority of the ontologies either own partial knowledge of multiple service categories, or only concentrate on a specific service delivery model. Only AoFeCSO and mOSAIC cover the entire cloud service models. Further, from the previous discussions, it can be found that AoFeCSO captures much more detailed specifications, including the in-depth cloud service OP and explicit cloud service entity relationship specifications, comprehensive DP and multi-sourced annotation specifications. Accordingly, these suggest that the proposed AoFeCSO owns competent domain coverage.

7.1.4.3 Quality of *Modelling*

Ontology modelling quality is often assessed based on the syntactic, structural and semantic quality aspects [21], where the logical consistency should be guaranteed. AoFeCSO was initially built by using Protégé. This means it follows formal OWL2 syntactic features for every axiom assertions. It adopts ReasoningOP, which enables it to reason new cloud service specifications such as inferred membership functions, property constraints and other object relationships. Its DL consistency has been verified (by HermiT and FaCT++) whenever any new information is added. Consequently, the modelling quality of AoFeCSO is kept to the latest standard.

7.1.4.4 Suitability for Service Retrieval and Recommendation Tasks

For the suitability evaluation, it compares AoFeCSO with other existing service specification models for service retrieval and recommendation tasks.

Regarding the suitability of the service recommendation tasks, the proposed approach is found to be advanced in three main aspects (refer to Table 7.2): I) It facilitates a user-friendly recommendation process due to the comprehensive keywords annotation presentation, whilst this assistance feature is seldom available in other service recommendation tools. II) It is by far the first tool that provides comprehensive service recommendation functions for diverse service delivery models and categories. III) The recommendation functions consider the fuzziness occurred in cloud service specifications; this enables a clearer view of

Table 7.2 Service Attributes Processing: Service Recommendations

Cloud service recommendations	**Other existing practices**	**AoFeCSO & CSR**
Description/explanation of the keywords	Few, partially, single source [150, 122, 128]	Full, multiple sources
Cross/multiple service categories/models	Partial [7, 59]	Yes
Fuzzy cloud specifications considered	N/A	Yes; processed during the recommendation process and represented in the recommendation ratios

Table 7.3 Overall Service Attributes Processing Effectiveness

Overall effectiveness comparison	**Other models and service recommendation systems**	**AoFeCSO & CSR**
Description of service attribute	Yes [7, 122, 144, 161, 73, 121, 59, 90, 151, 60]	Yes
Granular service attribute details	Very few [7]	Yes
Service attribute connections	N/A	Yes
Service attribute fuzziness specification	N/A	Yes, through collaborative fuzzy weight rating
Service/provider relationships	N/A	Yes

the small differences between similar services through more precise service recommendation ratios.

Additionally, as Table 7.3 summarises, the proposed approach is able to capture and present extended service specifications from a variety of aspects, e.g. showing multiple service model information, explaining granular details of service attributes, revealing service attribute connections, and processing fuzzy service specifications. Fundamentally, it is argued that other existing work is held back by their conventional inflexible ontology definition and implementation, whereas our approach rests on a loosely-coupled class and relation hierarchy.

As a result, seen from the above case study and comparison data, the proposed approach offers distinguished effectiveness for cloud specification processing with regard to the full range service recommendation and retrieval tasks.

7.1.4.5 Adoption and Use

In addition to the present use, AoFeCSO is also actively involved in a number of research projects. Indeed, its knowledge is being widely used for recent service brokerage [41] and optimisation [43] studies. While being adopted to assist service optimisation tasks, it can provide adequate semantic support to compare cloud services with similar functions, features, characteristics, etc. Further, as being used for service brokerage tasks, it would greatly enhance service matchmaking for cloud (resource) interoperability enablement. Indeed, the comprehensive service specifications across multiple abstraction layers make it a preferred knowledge for a wide range of service selection-relevant tasks.

7.2 Cloud Service Operation Specification and Execution

According to the native cloud service API reference documents, various service operations can be comprehensively specified using SAMOS framework.

7.2.1 Specification of IaaS Service Operations

In CC, IaaS services are generally provisioned to fulfil various computing resource needs for different users. Among all of the resources, IaaS compute service is regarded as a typical example that is widely consumed by many user types (e.g. individuals, companies, organisations). Indeed, the majority of such services offer choices for a wide range of VM sizes, OSs, software bundles, etc. For available service management options, accordingly, there are usually various VM-oriented operations available, such as to create, access and manage the service instances (VMs). In fact, the options of these operations are found very similar among CSPs. This means that the specification patterns would appear to be similar for the involved entities, entity data type formats, and entity operational relationships. SAMOS can effectively reveal both the similarity and the uniqueness among service operations from distinct providers.

To demonstrate how SAMOS framework can be applied to real-life IaaS cloud services, a series of examples are provided using operations selected from two CSPs. The specifications given below are divided according to their execution levels, i.e. service level, CSI level and PSSA level respectively.

7.2.1.1 Typical Operations of an IaaS Compute Service

Table 7.4 AWS EC2 Service Level Operation Specification

Granular Service Operations	***AWS EC2***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
List VM Instance	SIR	Unconditional	EC2 Region(M)	EC2 InstanceIDs	Unconditional
Create VM Instance(s)	SMR	< account allowance, i.e. 20 instances per region	EC2 RequestCount(O), EC2 InstanceType(M), EC2 AMIID(M), EC2 KeyName (M), EC2 SecurityGroup(O), EC2 Region(M), EC2 Monitor(O), EC2 AvailabilityZone (O), etc.	EC2 InstanceID(s)	Instance(s) are in "running" state
Reboot VM Instances	SMR	Instances are in "running" state	EC2 InstanceIDs(M)	Operation Succeeded	Instances are in "running" state
Stop VM Instances	SMR	Instances are in "running" state	EC2 InstanceIDs(M)	Operation Succeeded	Instances are in "stop" state
Resize VM Instances	SMR	Instances are in "stop" state	EC2 InstanceIDs(M), EC2 InstanceTypes(M)	Operation Succeeded	Instances are in "stop" state
…					

Table 7.5 Rackspace Cloud Servers Service Operation Specification

Granular Service Operations	***Rackspace Cloud Servers***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
List VM Instances	SIR	Unconditional	Rackspace Region(M) Rackspace FlavorID(M)	Rackspace CloudServerIDs	Unconditional
Create VM Instance	SMR	< Rackspace Absolute CSLimits, i.e. 100	Rackspace Server name(M), Rackspace ImageRef(M), Rackspace OSDiskConfig (O), Rackspace Metadata(O), Rackspace KeyPair(O), etc.	Rackspace CloudServer InstanceID	Instance is in "ACTIVE" state
Reboot VM Instances	SMR	Unconditional	Rackspace CloudServerID(M), Rackspace RebootType(M), e.g. SOFT, HARD	Operation Succeeded	Instances are in "ACTIVE" state
StopVM Instances	X	X	X	X	X
Resize VM Instances	SMR	Instances are Rackspace Standard Flavor	Rackspace CloudServerID, Rackspace FlavorID(M)	Operation Succeeded	Unconditional
…					

Table 7.4 and 7.5 demonstrate some cloud service level operation specifications retrieved from CSAMO. These typical operations belong to two IaaS services, i.e. AWS EC2 [1] and Rackspace Cloud Servers [118]. In comparison, although the two services own some service operations in common, the specifications are very different from each other. For instance, they all

support retrieving owned instance through the “List Owned VM” request, which is seen a SIR and requires only a pre-selected region information as the mandatory SRParameter. Obviously, the two region parameters are two different PSSAs: despite the fact that AWS and Rackspace both have regions of the same geographic locations (e.g. UK and USA), the two entities represented them are known distinctively and therefore have their own strings (formats) of presentation (data type). On successful execution, both SIR executions would not alter the services, hence there would be no change to their SRPostCondition; the SROutcomes for the SIRs are seen as two series of ID lists of the owned service instances.

On the other hand, for SMR operations, EC2 offers more options than the other for the listed operations. I) While both services allow users to create new instance, the SRPreCondition and SRParameter in need are seen distinctive: For preconditions, EC2 uses a maximum of 20 running instances per region for ordinary users, whereas Rackspace limits the total instance count to 100 for all users. For SRParameter, EC2 requires a specific AWS region, instance type, AMI (VM image) ID and key name (for user authentication use) as mandatory parameters, plus security group (virtual firewall), request count (number to be created), availability zone (sub zones for the region), monitor (for frequent monitor), etc. as optional parameters; Rackspace Cloud Servers needs mandatory server name, Flavor (instance type) ID and ImageRef (VM image reference), as well optional parameters such as OSDiskConfig (disk configuration value), metadata (custom server metadata), key pair (for user authentication use), etc. As the request complete successfully, both would return the new created VM instance IDs as SROutcome II) Except the major distinctiveness which rests in Rackspace does not provide “stop” command for the VM instances, there are still clear differences between the two providers, even for the basic “reboot” and “resize” commands. Indeed, the reboot option offered by Rackspace accepts additional “SOFT/HARD” parameter for the respected reboot operations, whereas EC2 simply needs subject instance IDs;

For instance resize operations, EC2 needs the subjects to be at “stop” state whilst Rackspace requires the instance to be a “standard flavoured” VM.

7.2.1.2 Typical Operations of an IaaS Compute Service Instance

In addition, a number of typical IaaS service instance operation specifications are illustrated in Table 7.6. In contrast with the above service (class) request operations, IaaS service instances (individual) are often provisioned with more request options, due to the considerably more instance-specific SIR and SMR operations involved.

Considering SIR operations, for each service/instance attribute that is associated with the instance, there would be a respected SIR to retrieve the dynamic information, e.g. to get the instance’s architecture, type, public IP address, etc., as illustrated in Table. Generally speaking, these SIR operations requires very few to no more SRParameter other than the instance’s ID, and would return the respected SROutcome according to their expected data types. Such SIR would unlikely result into any changes to the instance.

Table 7.6 AWS EC2 Service Instance Operation Specification

Granular CSI Operations	***AWS EC2 Instance***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
Get VM Architecture	SIR	Unconditional	EC2 InstanceID(M)	EC2 Instance Architecture	Unconditional
Get VM Instance Type	SIR	Unconditional	EC2 InstanceID(M)	EC2 InstanceType	Unconditional
Get Instance PublicIP	SIR	Instance is in “running” state	EC2 InstanceID(M)	EC2 InstancePublic IP	Unconditional
Duplicate VM Instance	SMR	< account allowance, i.e. 20 instances per region	EC2 InstanceID(M) EC2 RequestCount(O), EC2 InstanceType(O), EC2 AMIID(O), EC2 KeyName (O), EC2 SecurityGroup(O), EC2 Monitor(O), etc.	EC2 InstanceID(s)	Instance(s) are in “running” state
Create VM Image	SMR	Unconditional	EC2 InstanceID(M)	EC2 AMIID	AMI is in “available” state
Terminate VM Instance	SMR	Instance is NOT in “terminated” state	EC2 InstanceID(M)	Operation Succeeded	Instance is in “terminated” state
…					

On the other hand, there are several SMR operations are seen applicable only for a specific IaaS service instance, such as to create the instance's image and to duplicate the instance. Take "Duplicate Instance" for example, the SMR has very similar SRPreCondition and SRPostCondition with EC2 service's "Create Instance(s)" operation. This is due to the very same fundamental API request they both are mapped to. Take EC2 instance as an example, the "Create Image" operation is to save the latest snapshot of the VM and then create an image of it (for duplication, records, backup, etc. purposes). The SMR can be initiated regardless of the instance's status, and therefore, requires no SRPreCondition. On after successful execution, the created image's AMI ID is returned as the SROutcome, with the SRPostCondition of the AMI is at "available" state. Except such service instance-specific operations, the rest are seen as the singular version of the IaaS service SMR operations, i.e, "Start VM", "Stop VM", "Terminate VM", as long as they are of the same manipulation function as for the service. Obviously, the SRPreCondition, SRParameter, SROutcome and SRPostCondition of such operations would also be transformed be for the instance only.

7.2.1.3 Typical Operations of an IaaS Compute Provider-specific Entity

Additionally, to provide comprehensive functionalities, IaaS service console often comprise a series of additional concepts and entities management functions that are specifically related to certain aspects of the service, i.e. entities representing certain computational concepts, resource pools, resource interfaces, etc. For these PSSA entities, many are supplied with certain additional management operations. Indeed, these additional service operations add up to the range of service instance configuration tasks. Similarly to the nature of PSSA, most PSSA operations are recognisable only for a certain single CSP.

Table 7.7 AWS EC2 Provider-Specific Entity Operation Specification

Granular PSSA Operations	***AWS EC2 AMI (VM image)***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
Get Image Name	SIR	Unconditional	EC2 AMIID(M)	EC2 AMIName	Unconditional
Get Image Platform	SIR	Unconditional	EC2 AMIID(M)	EC2 Instance Platform	Unconditional
Create VM Instance(s)	SMR	< account allowance, i.e. 20 instances per region	EC2 InstanceID(M) EC2 RequestCount(M), EC2 InstanceType(M), EC2 AMIID(M), EC2 KeyName (O), EC2 SecurityGroup(O), EC2 Monitor(O), etc.	EC2 InstanceID(s)	Image is in "available" state
Delete Image	SMR	Image is in "available" state	EC2 AMIID(M)	Operation Succeeded	Unconditional
…					

Table 7.7 illustrates some operation specification data of EC2 AMI. As VM image is one typical IaaS service entity that applies to all IaaS VM services, some of the entity operations may still be reused for other IaaS providers, e.g. Create VM Instance(s) and Delete Image are known as two general operations which are supported by almost all IaaS VM service providers. However, for SMR operations like the above, there would be very distinct condition and parameter requirements between different CSPs. For instance, for EC2 AMI, the SRPrecondition and SRParameter are found similar to which for EC2 instance' Duplicate Instance operation and EC2 create VM Instance(s) operation (also mapped to the same API call). Additionally, the combination of SIR operations of these provider-specific entities would mostly vary from distinct providers. There are few chances of compatible cross-provider SROutcome entities even for the same operation, except the cases such as some service providers have strong industrial relationships with each other, some providers employ (rely) other's service resources, etc.

7.2.2 Specification of SaaS Service Operations

In contrast with IaaS and PaaS services which serve fairly limited number of purposes, SaaS services are usually found in a diversity of functions, e.g. business applications such as CRM, ERP, accounting software services. Due to the variations and complexity of the functions, the available service, service instance and SaaS provider-specific operations would vary dramatically among distinct service types. In fact, for some SaaS services, there may be no applicable service instances. For instance, online storage services (e.g. Google Drive [56]) would only have some provider-specific SaaS entities (e.g. the file nodes, the “Bin”, the user account). Therefore, SaaS service operation specification patterns are seen diverse for each specific software function category. The example used here is cloud load balancer services. In contrast with others software functions, the load balancer application provide a moderate view considering the overall functional operations available, service entity constitution, as well as a layered entity reference architecture.

7.2.2.1 Example Operations of a SaaS Cloud Load Balancer Service

Using GoGrid Dynamic Load Balancers (GDLB) and Rackspace Cloud Load Balancers (RCLB) as examples, Table 7.8 and 7.9 list the specifications of some typical service level load balancer operations. For both services, there are general operations such as to obtain the owned service instances and create new instance, similarly as for ordinary IaaS and PaaS services. Nevertheless, compared with services from other delivery models, the total number of such operations is typically smaller. This is due to the fact that these services tend to be simpler and generally have fewer interactions with other services and service entities.

Table 7.8 GoGrid Dynamic Load Balancers Service Operations

Granular Service Operations	***GoGrid Dynamic Load Balancers***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
List Load Balancer Names	SIR	Unconditional	GoGrid Datacenter(M)	GoGrid LoadBalancer InstanceNames	Unconditional
List Load Balancer Instance Addresses	SIR	Unconditional	GoGrid Datacenter(M)	GoGrid LoadBalancer VIPs	Unconditional
Create Load Balancer Instance	SMR	Uncnditional	GoGrid Datacenter(M), LoadBalancerName(M), GoGridRealIP(M), GoGridVIP(M), GoGrid BalancerAlgorithm(M)	GoGrid LoadBalancer InstanceID	Load Balancer instance is in "ON" state
Delete Load Balancers	SMR	Load Balancer is NOT in "UPDATING" state	GoGrid LoadBalancerInstanceID(M)	Operation Succeeded	Load Balancer Instance is NOT in "UNKNOWN" state
…					

Table 7.9 Rackspace Cloud Load Balancers Service Level Operation Specification

Granular Service Operations	***Rackspace Cloud Load Balancers***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
List Load Balancer Instance Names	SIR	Unconditional	Rackspace Region(M)	Rackspace Cloud LoadBalancer InstanceNames	Unconditional
List Load Balancer Instance Addresses	SIR	Unconditional	Rackspace Region(M)	Rackspace Cloud LoadBalancer Addresses	Unconditional
Create Load Balancer Instance	SMR	< Rackspace Absolute LBLimits, i.e. 25	Rackspace Region(M), LoadBalancerName(M), LoadBalancerPort(M), Rackspace CloudServer(O),Rackspace CloudLoadBalancer ExternalNode(O), Rackspace VirtualIP(M) ,etc	Rackspace Cloud LoadBalancer InstanceID	Load Balancer Instance is in "ACTIVE" state
Delete Load Balancer	SMR	Load Balancer is NOT in "UPDATING" state	Rackspace CloudLoadBalancer InstanceID(M)	Operation Succeeded	Load Balancer Instance is in "ACTIVE" state
…					

More specifically, considering the SIR operations, any cloud load balancer instance created can be allocated with a public IP address; to retrieve such information, there are SIRs such as "List Load Balancer Instances" and "List

Load Balancer Addresses". In addition, both services would offer the same SIR operations as listed, for which the SRPreCondition, SRPremeter, SROutcome and SRPostCondition specifications appear to be similar, with only differences of the entities involved (owning by the respected providers).

In the meantime, for the listed SMR operations, there is also similarity over the load balancer instance control options between the two services. It is found that some of the SMR can only be applied to a single instance subject at a time, for the reason that it is very unlikely for users to perform massive management operations simultaneously for multiple load balancer instances (e.g. create multiple load balancer instances at once).

7.2.2.2 Example Operations of a SaaS Cloud Load Balancer Service Instance

SaaS cloud load balancer services do offer a series of operations at the service instance level, seen as the relevant load balancer instance configuration tasks. As illustrated in Table 7.10, there are many SIR and SMR operations available for an individual load balancer (RCLB).

Table 7.10 Rackspace Cloud Load Balancers Service Instance Operations

Rackspace Cloud Load Balancer Instance	**Operation Type**	**Granular Service Instance Operations (partial)**
Load balancer instance general tasks	SIR	Get LoadBalancer InstanceName, Get LoadBalancer InstanceStatus, Get LoadBalancer InstanceAddress
	SMR	Edit LoadBalancerName
Nodes configuration tasks	SIR	List LoadBalancer InstanceNodes, List LoadBalancer InstanceNodeAddresses
	SMR	Add LoadBalancer Instance Nodes, Delete LoadBalancer Instance Nodes
General load balancing management tasks	SIR	Get LoadBalancing Algorithm , Get Load Balancing Port, Get LoadBalancing AccessRules
	SMR	Edit LoadBalancing Algorithm, Edit Load Balancing Port, Add LoadBalancing AccessRules, Edit LoadBalancing AccessRules, Delete LoadBalancing AccessRules
Rackspace exclusive feature management tasks	SIR	Get RackspaceCloudLoadBalancer HealthMonitor, Get RackspaceCloudLoadBalancer SessionPersistence, Get RackspaceCloudLoadBalancer ConnectionThrottling, Get RackspaceCloudLoadBalancer ErrorPage, Get RackspaceCloudLoadBalancer Logging
	SMR	Edit RackspaceCloudLoadBalancer HealthMonitor, Edit RackspaceCloudLoadBalancer SessionPersistence, Edit RackspaceCloudLoadBalancer ConnectionThrottling, Edit RackspaceCloudLoadBalancer ErrorPage Edit RackspaceCloudLoadBalancer Logging, etc.

Table 7.11 Rackspace Cloud Load Balancers Service Instance Operation Specification

Granular CSI Operations	***Rackspace Cloud Load Balancer Instance***				
	Type	**SRPre Condition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
Get Load Balancing Algorithm	SIR	Load Balancer is in "ACTIVE" state	Rackspace Cloud LoadBalancer InstanceID(M)	Rackspace LoadBalancing Algorithm	Unconditional
List Load Balancer Instance Nodes	SIR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer InstanceID(M)	Rackspace Cloud LoadBalancer Instance NodeID(s)	Unconditional
Edit Load Balancer Instance Health Monitor	SMR	Load Balancer is in "ACTIVE" state	Rackspace Cloud LoadBalancerInstanceID(M), Rackspace Cloud LoadBalancer HealthMonitor(M)	Operation Succeeded	Load Balancer is in "Active" state
Add Load Balancing Access Rule	SMR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer Instance ID(M), Rackspace CloudLoadBalancing AccessRule (M)	Operation Succeeded	Load Balancer is in "ACTIVE" state
Add Load Balancer Instance Nodes	SMR	Load Balancer is in "ACTIVE" state	Rackspace CloudServer(O), Rackspace CloudLoadBalancer ExternalNode(O), Rackspace CloudLoadBalancer InstanceNodePort(O), etc.	Operation Succeeded	Load Balancer is in "ACTIVE" state
Delete Load Balancer Instance Nodes	SMR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer InstanceNodeIDs(M)	Operation Succeeded	Load Balancer is in "ACTIVE" state
…					

More specifically, Table 7.11 shows the detailed information of a series of RCLB service operations. For SIR, except general information such as the name, ID, status, etc. that is widely available for all the service providers, RCLB offers additional functionalities such as a series of advanced load balancing algorithms (e.g. "weighted round robin" and "weighted least connections"), load balancer health monitor and access rules. While requesting the information, the SIR SRPreConditions require the load balancer instance to be at the "Active" state. Additionally, as a cloud load balancer typically consists of a series of nodes, "List Load Balancer Instance Nodes" action is then enabled. Due to the fact that RCLB supports both external (e.g. a public IP address) and internal (a private IP address or a VM instance in Rackspace cloud) nodes for load balancing tasks, the SROutcomes of the operations return only the instance

node IDs. By using the ID, additional provider-specific entity operations can be launched.

The load Balancer's SMR operations provide comprehensive configurations regarding the load balancing algorithm, access rule, health monitoring, logging, connection throttling, and session persistent, etc. management tasks. In fact, except for few SMRs which are similar regardless of any provider (e.g. those load balancer instance node management operations), the majority of the features are seen only applicable for RCLB. Hence, except some load baling ports and algorithms which might be recognisable for other load balancer services, the entities involved in the SMRs' SRParameters are meaningless to all other services, even for those owned by Rackspace.

7.2.2.3 Example Operations of a SaaS Cloud Load Balancer Provider-specific Entity

At the PSSA level, a number of additional load balancer service operations are usually presented for certain granular service access and controls. Such as routing, logging and load balancer node management tasks (see Table 7.12). Here, cloud load balancer node is seen as a typical entity that applies to all of such services. For the majority of such CSPs, it usually comes with some a series of SIR and SMR operations.

As the fundamental element of load balancer instances, each node is normally given a unique ID. From the ID, the address information can be obtained: in case of an external node, it would point to a public IP Address; a private node would either lead to a private IP address or a RCS instance ID. Additionally, RCLB also allow users to edit the conditions of the node from "Enabled, Disabled or Draining Connections". Under a weighted load balancing algorithm, each nodes presented in the instance is associated with a weight (an integer of 1-100); the load balancer instance would distribute the traffic based on the proportional relationships among the weights.

Table 7.12 Rackspace Cloud Load Balancers Provider-Specific Operation Specification

Granular Service Instance Node Operations	***Rackspace Cloud Load Balancer Instance Node***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SRPostCondition**
Get LoadBalancer Instance Node IP	SIR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer InstanceNodeID(M)	Rackspace Cloud LoadBalancer InstanceNodeIP	Unconditional
Get LoadBalancer Instance Node Status	SIR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer InstanceNodeID(M)	Rackspace Cloud LoadBalancer InstanceNode Condition	Unconditional
Get LoadBalancer Instance Node Port	SIR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer InstanceNodeID(M)	Rackspace Cloud LoadBalancer InstanceNode Port	Unconditional
Edit LoadBalancer Instance Node Weight	SMR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer InstanceNodeID(M), Rackspace CloudLoadBalancer InstanceNodeWeight(M)	Operation Succeeded	Load Balancer is in "ACTIVE" state
Edit LoadBalancer Instance Node Status	SMR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer InstanceNodeID(M), RackspaceCloud LoadBalancerInstance NodeCondition(M)	Operation Succeeded	Load Balancer is in "ACTIVE" state
Delete Load Balancer Instance Node	SMR	Load Balancer is in "ACTIVE" state	Rackspace CloudLoadBalancer InstanceNodeID(M)	Operation Succeeded	Load Balancer is in "ACTIVE" state
…					

7.2.3 The Unified Interface for Real-world Cloud Service Access and Manipulation Tasks

The above cloud service specification case studies suggest that SAMOS framework can adequately model and specify the variety of service operations from distinct service types, delivery models and provider clouds. Based on these specifications, USAMS prototype is implemented to enable a unified interface for comprehensive cloud service management tasks. This section demonstrates some case studies on cloud service access and manipulation tasks by utilising a real-life IaaS service (AWS EC2).

USAMS adopts a structured interface for cloud service operation retrieval preparation and execution. Basically, all cloud services, CSIs and PSSAs are initially displayed in a small panel. Seen in Figure 7.10, the panel comprises four buttons: “Description”, “Use Entity”, “Information” and “Manipulation”. By clicking the “Description”, users can view its annotation description through platform sub system interactions. The “Information” and “Manipulation” buttons lead to the respected SIR and SMR operations, which are retrieved dynamically from CSAMO. Then, if a user’s API account authorisation permits, one can execute the respected operations via the interface.

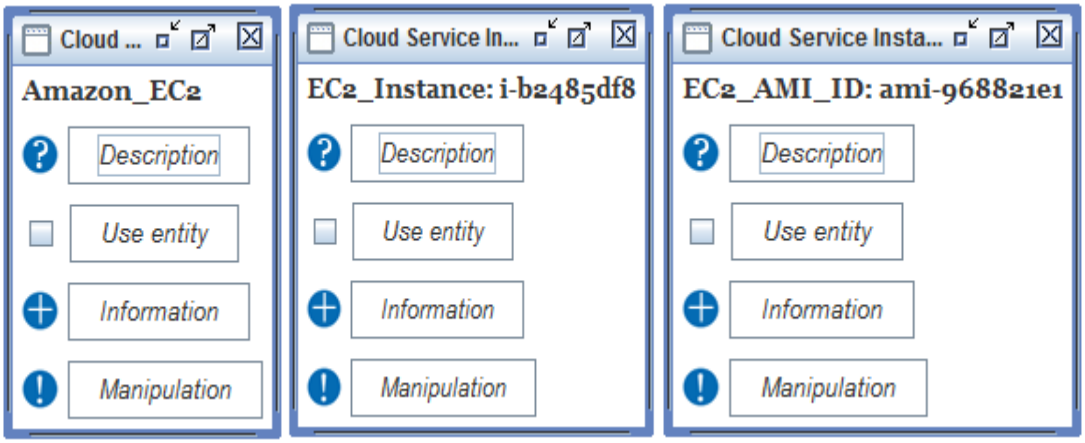


Figure 7.10 Initial Cloud Service Entity Panels

7.2.3.1 Cloud Service Access Operations

Using AWS EC2 as the example cloud service, the following contents demonstrate the processes of the SIR operations retrieval, followed by real-time service and service instance accesses tasks. To illustrate the practical service access example, Figure 7.11 demonstrates the appearance of cloud service SIR operation retrieval. In fact, as the “Information” button is clicked, a dynamic ontology lookup is performed, where the relevant SIR operations specifications are extracted, processed and displayed for further commands. As a typical IaaS compute service, EC2 is provisioned with various service operations which are closely relevant to VM configuration tasks: sizes, access keys, security groups, regions, etc. Further, while acquiring the SIR option lists, each operation would

be mapped with a specific API call for execution preparation. Once the preparation is ready, a “Request” button will be presented next the operations, notifying the user that one can request the information through pre mapped service programming calls.

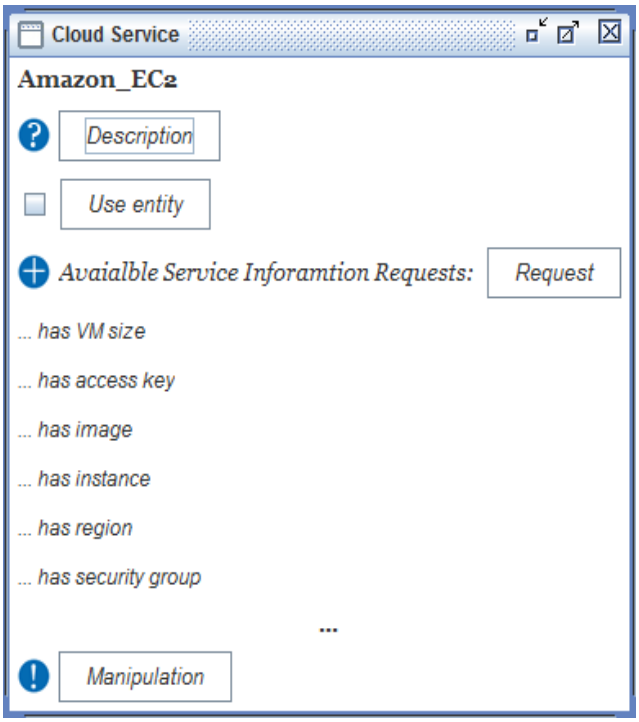


Figure 7.11 EC2 SIR Operations Retrieval

Then, as the user clicks the “Request” button, relevant pre-mapped API calls are dynamically initiated. If the requests successfully execute, the respected service data is obtained from the service provider (i.e. AWS EC2) via the API requests (See Figure 7.12). For instance, EC2 involves several aspects that are relevant to the usage of VMs: VM sizes, images, security groups, regions, etc. All of the information can be acquired via the SIR execution. Additionally, via the “has instance” operation, the user can retrieve the VM instances owned for a specific region.

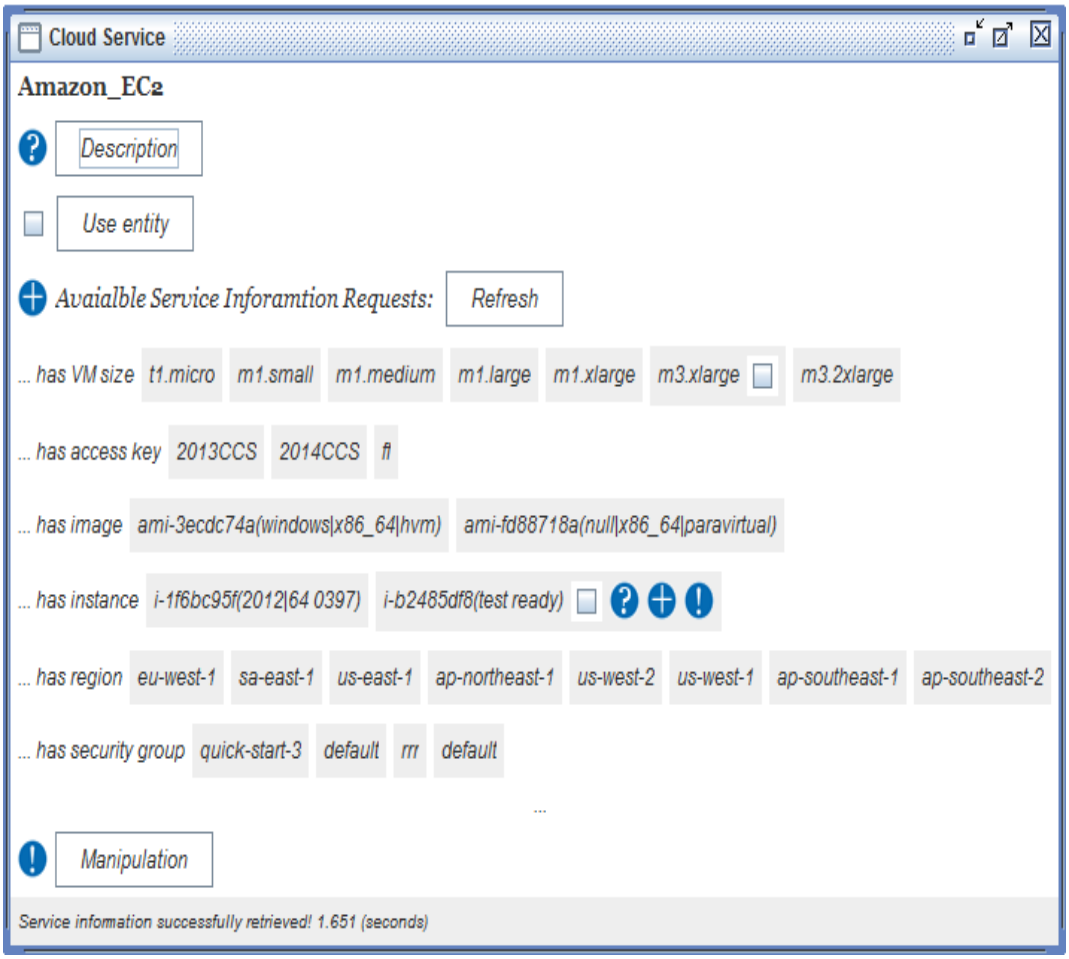


Figure 7.12 EC2 SIR Operation with Real-time Cloud Data Access

Subsequently, for the dynamically retrieved service information, there can be additional actions, if the entity type allows. As previously discussed, all CSIs and PSSAs are backed by subsequent actions which are uniquely presented, owning to their specific characteristics and usages. As shown in Figure 7.13, the “i-b2485df8” instance provides an example of SIR interaction across service and CSI levels. While the instance’s SIR operations successfully execute, the instance-specific information is then acquired from EC2. Later, based on the entity type of the newly obtained instance data (entity), they may also lead to their own lists of service request operations.

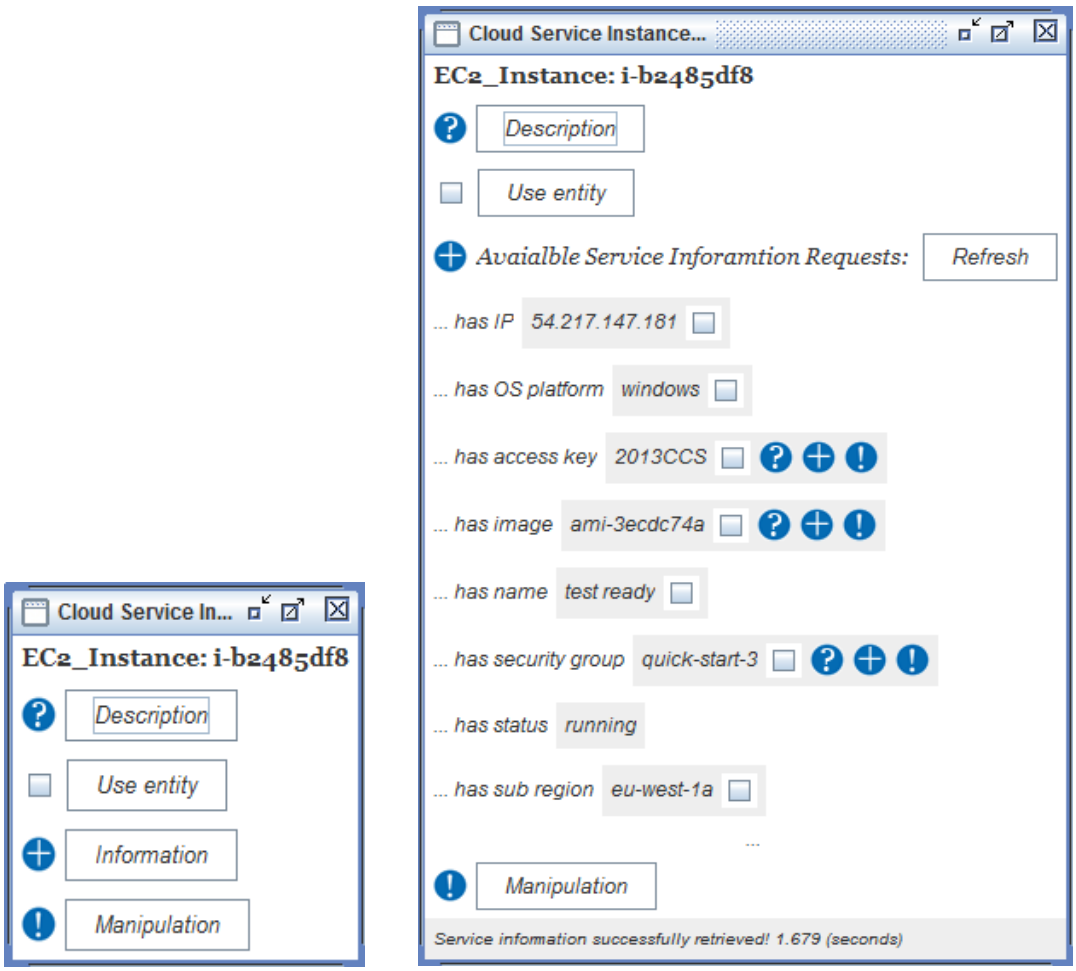


Figure 7.13 EC2 Instance Cloud Service SIR Interactions

In the meantime, another use of the dynamically retrieved service information (SROutcome) is known entity reuse. In this example, except the “running” status of the EC2 instance (which is unusable), all other information can be selected as either SRSubject or SRParameter or both for relevant service operations (e.g. the IP address can be used for another service; the access key can be used to create another instance).

7.2.3.2 Cloud Service Manipulation Operations

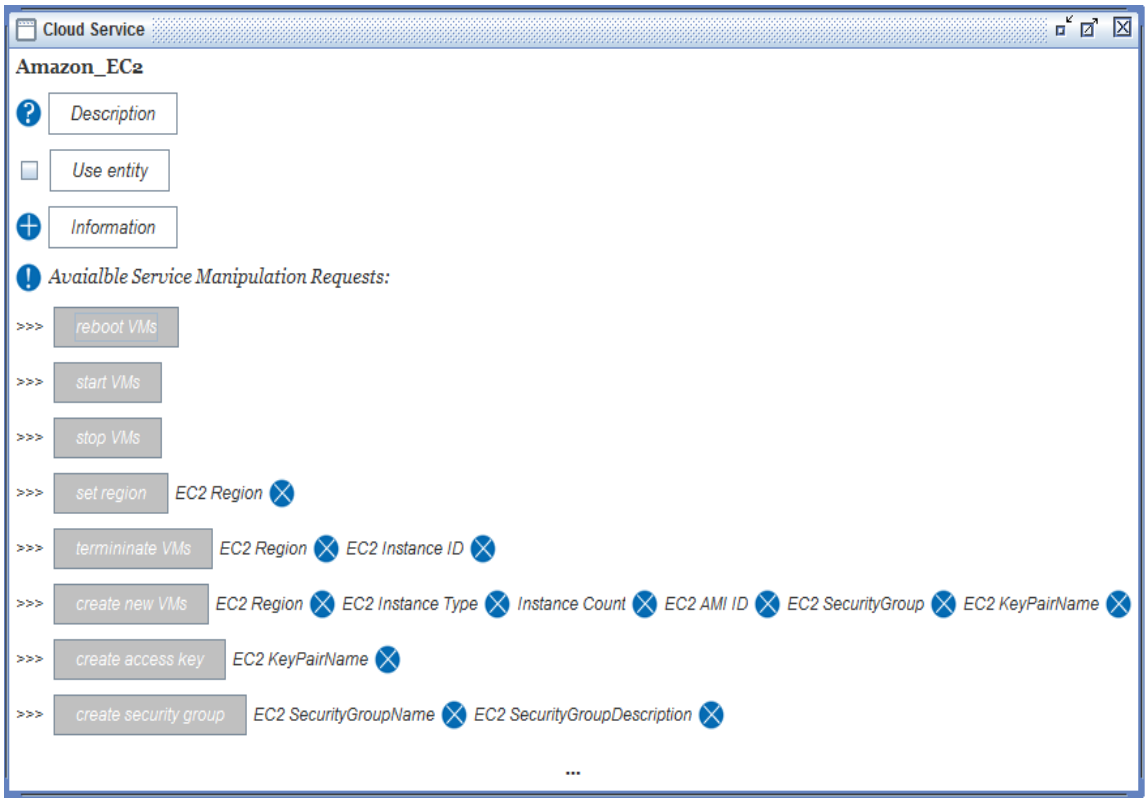


Figure 7.14 EC2 SMR Operation Retrieval

Continuing with the above example IaaS service, SMR operations of EC2 can also be retrieved from the operation specification ontology. Displayed in Figure 7.14, this involves a series of general service instance configuration operations (e.g. create, start, stop and terminate), plus some management operations for its unique PSSA entities (e.g. EC2 KeyPair, SecurityGroup). As a user clicks at an operation command, an ontology look up process would be triggered, where the respected SRParameter requirements will be obtained and displayed along with the SMR. Shown in Figure 7.14, for instance, “set region” command would need the user to specify a relevant “EC2 Region”; “create security group” would require an input of “EC2 Security Group Name” and “EC2 Security Group Description”.

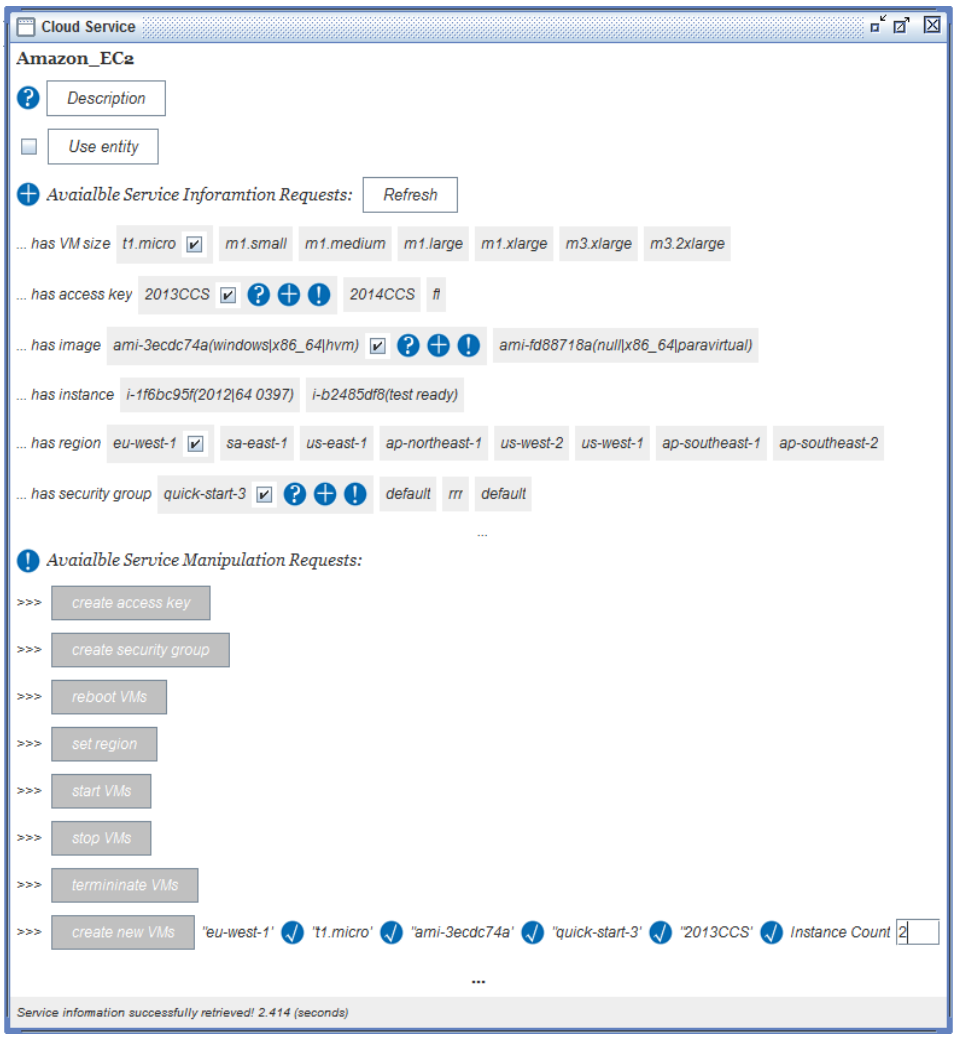


Figure 7.15 EC2 SMR Operation Preparation

To further explain how SMR operation is prepared and executed, a detailed example is given, using the “create new VMs” operation. The example SMR requires a number of parameters including “EC2 Region”, “EC2 Instance Type”, “EC2 AMI ID”, etc. As illustrated in Figure 7.15, these parameters can be entered through either selection of the previously obtained service information, or manual typed input. As the system detects user’s input, fulfilled parameters would fit into the respected position whilst the icon followed which would transit from “unchecked” into “checked”. By the time all of the SRParameters are fulfilled, a dynamic service condition checking process is initiated prior to the operation execution. In this case, the command requires the user to own no more than 20 instances per region.

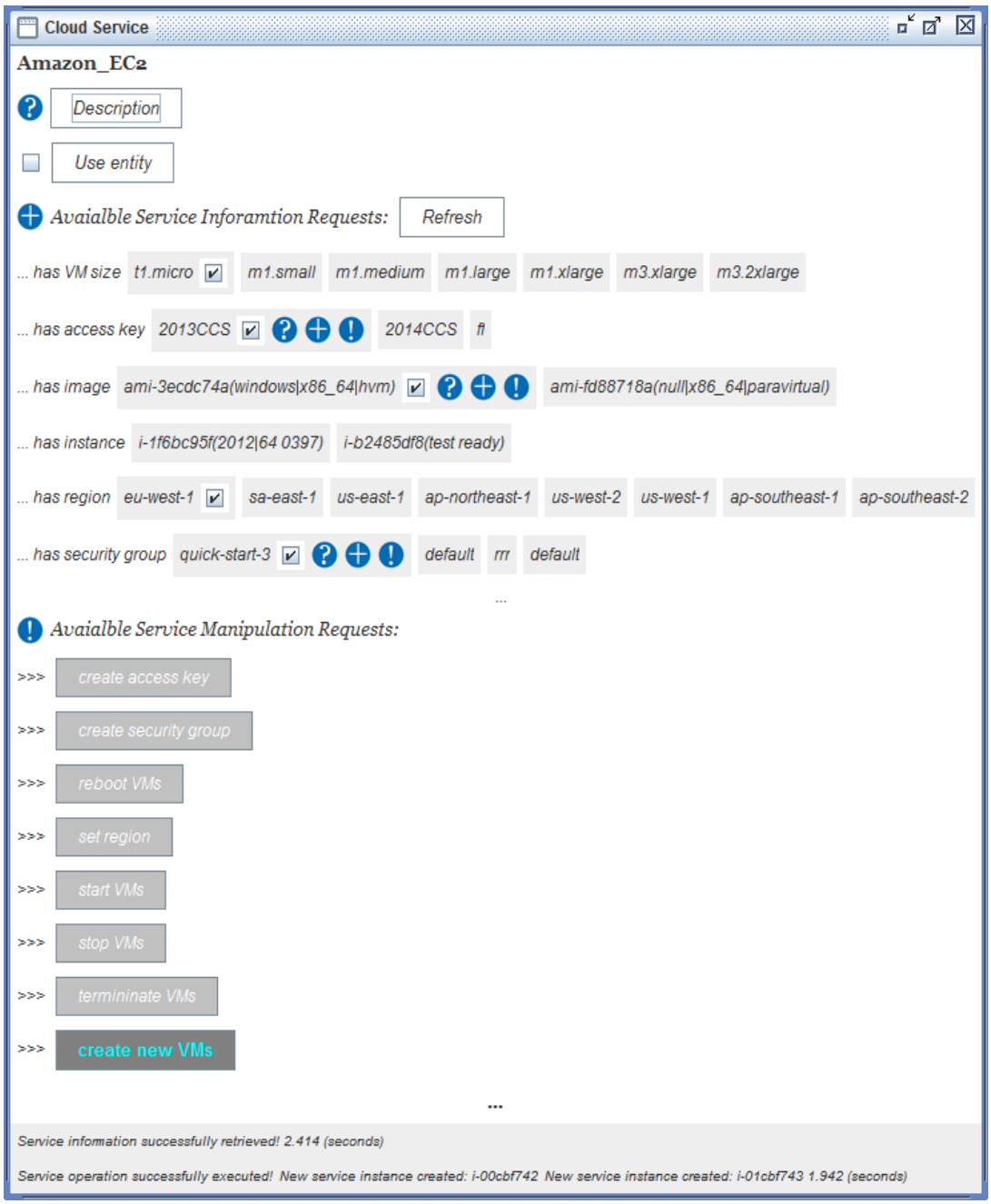


Figure 7.16 EC2 SMR Execution

Then, as all defined requirements are fulfilled, when the user clicks the SMR command, the pre-mapped API request is sent to the service provider. After some time, relevant respond will be returned from the provider, notifying the execution status. As shown in Figure 7.16, the request has executed successfully and has resulted two newly created VM instances: “i-00cbf742”

and “i-01cbf743”, which can be found at the bottom the panel. Subsequently, seen in Figure 7.17, the two new instances can be found through SIR updates.

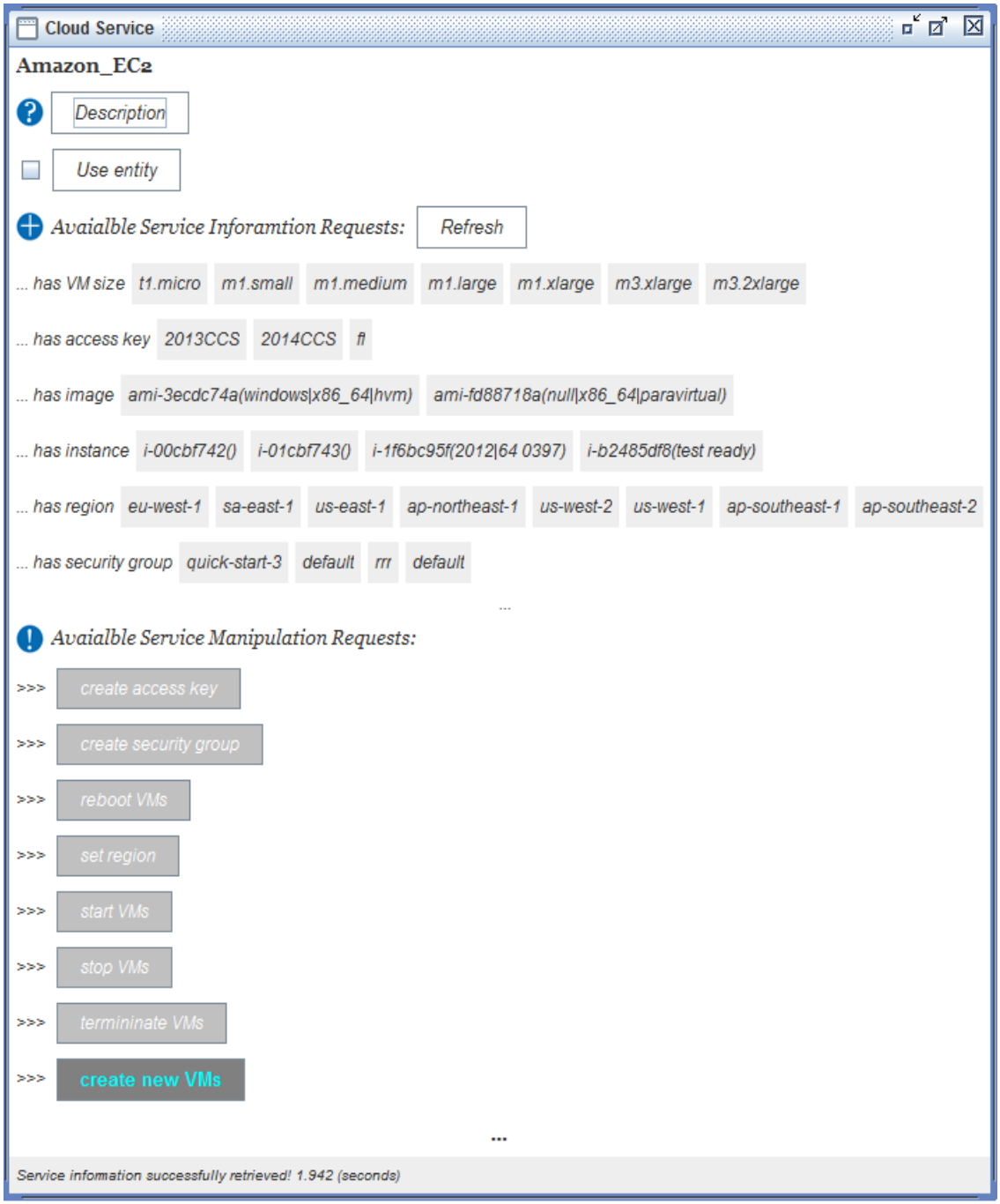


Figure 7.17 EC2 SIR Update After SMR Execution

7.2.4 Cloud Service Operation Assistance and Dynamic Orchestration

7.2.4.1 Cloud Service Operation Assistance - BASR

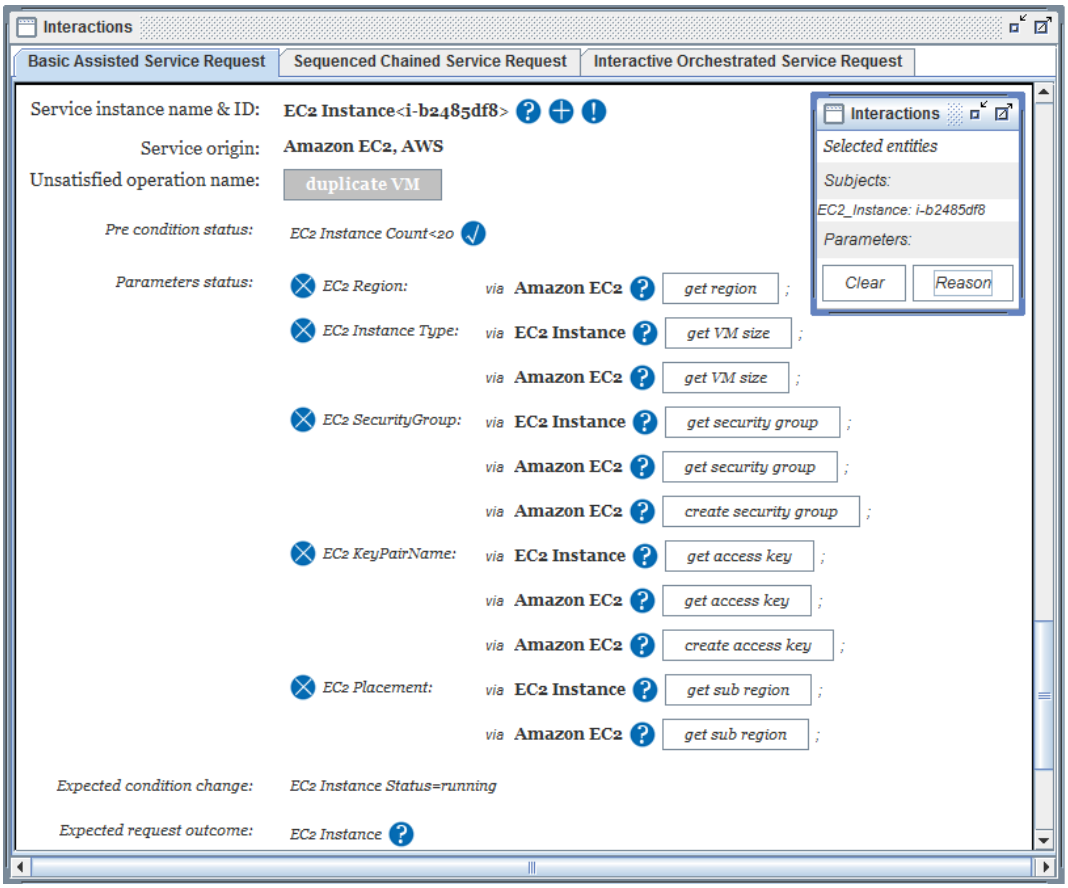


Figure 7.18 BASR Reasoning Assistance Example

As a means towards versatile SMR assistances for real-world cloud services, BASR tends to extract various types of information that can be relevant while preparing service operations' requirements. Figure 7.18 shows the reasoned outcome for the selected subject (an AWS EC2 instance with ID of "i-1f6bc95f"). Using the SMR "duplicate VM" as an example, the full details regarding the operation are displayed in the interface: the SMR requires a precondition of "EC2 Instance count < 20", which is currently satisfied for the user. There are five mandatory parameter needed, involving region, instance type, security group, etc.; as none of these are present, lists of information regarding how they

can be obtained are generated. For instance, “EC2 Instance Type” can be retrieved through the “get VM size” request launched on EC2 service or instance level; “EC2 Security Group” can be obtained by requesting existing ones or creating new ones.

In the meantime, BASR also reveals the expected changes for each SMR execution. In the example, if the SMR successfully executed, a duplicate instance will be created, which is seen as the outcome of the operation. Furthermore, considering the post-condition status after execution, the most obvious condition change is that the raised total owned instance number in contrast with before. Yet, due to the fact that the former status is less “meaningful” compared with the latter, the main condition change for the SMR is recorded as the new instance’s “running” state. More specifically, with the new instance’s running state, various types of instance manipulation (deployments, configurations, etc.) actions would be reasoned; in contrast, given the similar instance count-relevant condition, the possible following operations would still be reasoned as instance creation-related actions, which would be relatively unlikely to be implemented.

7.2.4.2 Cloud Service Operation Assistance - CCSR

CCSR enables serves to perform multiple service manipulation tasks simultaneously over the selected cloud service subjects. It enables users to deploy a combination of service manipulation tasks simultaneously, without having to go through different provider clouds and perform each individually. With appropriate parameters and dynamic service (instance) condition checks, the reasoning engine gathers similar SMR operations for which all requirements are satisfied. Figure 7.19 illustrates two CCSR operation examples produced based on the real-time selected service instance subject, parameters and condition check results. The reasoning is implemented with a number of subjects only, without any parameters: the subjects are seen as a number of “Rackspace Cloud Server Instance” and two “EC2 Instance”.

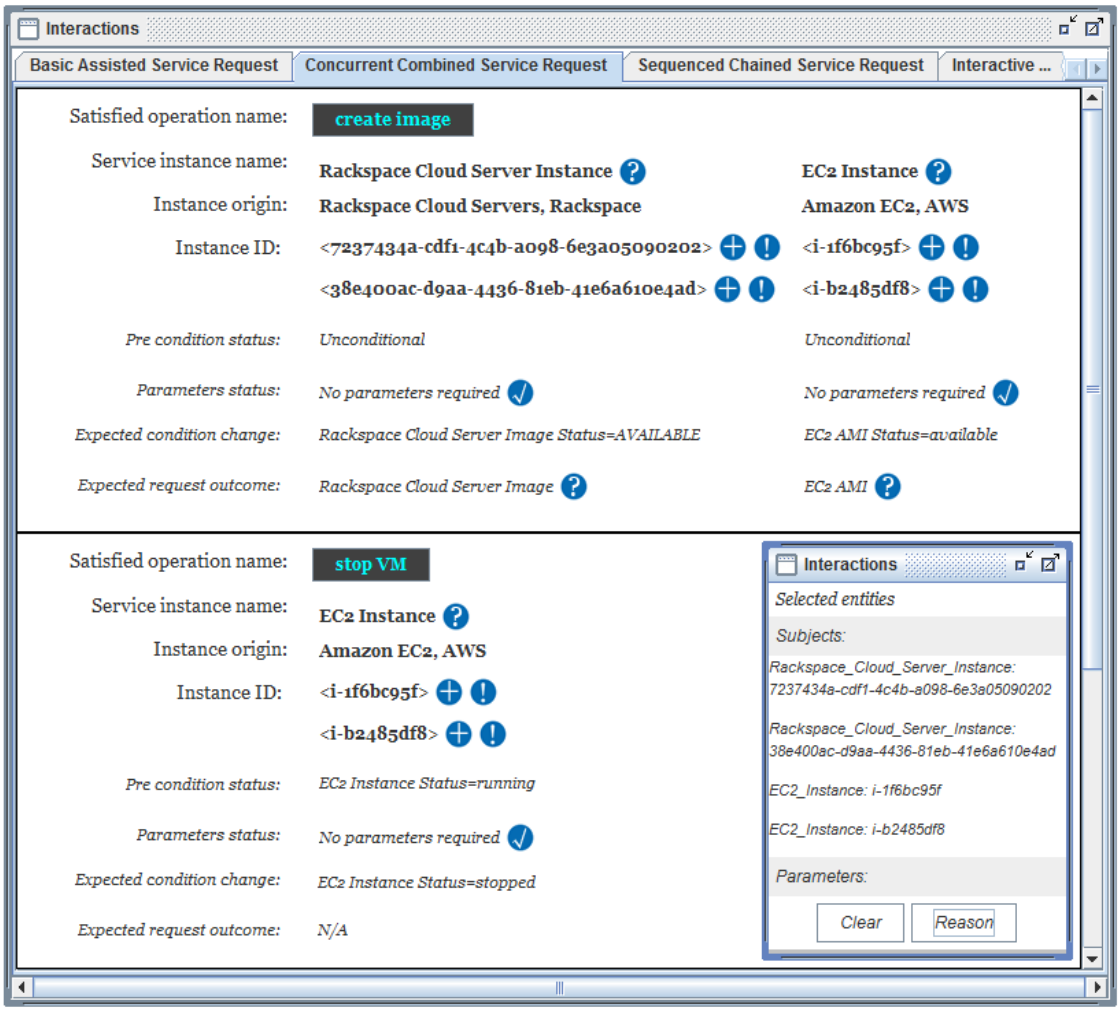


Figure 7.19 CCSR Reasoning Assistance Example

Seen in Figure 7.19, the first CCSR option reasoned, “Create image”, is produced under the following conditions: the SRPreCondition of the SMR is “unconditional”, which means it needs no precondition. Besides, the operation requires a subject, which is the ID of a VM; no additional parameter is required. As this is a “common” IaaS operation, such can be implemented despite distinct service providers. Then, if the requests are successfully executed, there will be a number of VM images produced in each provider cloud whilst the condition changes are known as the new images’ “available” state.

On the other hand, the second reasoned CCSR option is “stop VM”. Similarly as the above operation, it needs only some service instance as subjects no and no other parameters, whereas it can also be initiated for multiple distinct service

providers. However, due to the fact that the SMR requires a precondition of “instance status =running” and only those from AWS satisfy the state at that time, the CCSR is only deployable for the EC2 instances presented. Lastly, the expected condition change is known as the “stopped” state of the instances.

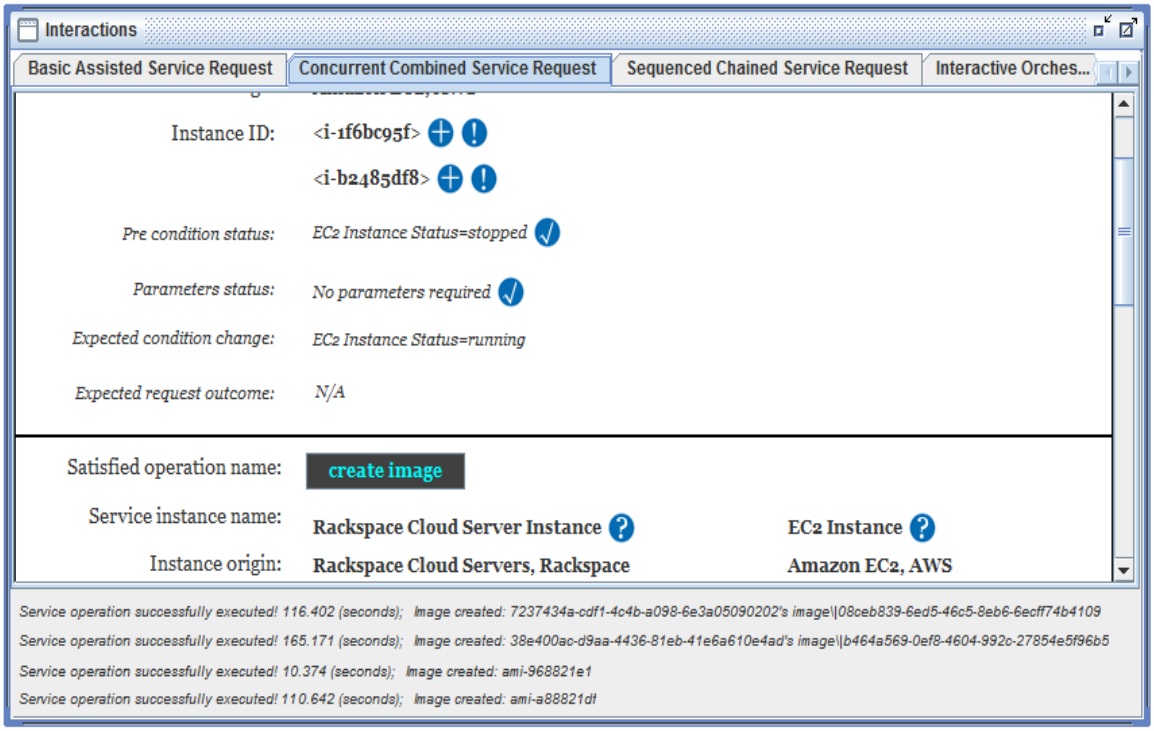


Figure 7.20 CCSR Reasoning Assistance Operation Execution

Then, Figure 7.20 demonstrates the outcome of a successful deployment of the “create image” CCSR command, for the selected AWS EC2 and Rackspace Cloud Server Instances (the whole process is also demonstrated via the native web page screenshots in Appendix C.i). Seen from the displayed messages, the concurrently executed operations compose four individual requests for each subject presented. After the executed, four new VM images are produced in the two provider clouds. From the dynamically retrieved image IDs, it can be seen that they follow the distinct patterns for the respected provider. In the meantime, the elapsed time for the instance image creation varies due to many factors including the size of the virtual disks, the operating system of the VM, a series of virtualisation platform and provider-specific aspects, plus some uncontrollable influences such as the real-time load and traffic of the clouds, etc.

7.2.4.3 Cloud Service Operation Assistance - SCSR

The intent of SCSR reasoning is to present a list of chained service operations, where each operation chain can be executed in a certain applicable sequence for a desired aim(s). This enables users to implement a process of service manipulations effectively, so that they do not need to manually operate them one another and wait for their completions. For a selected operation chain, once the user initiates the very first action command, the system would deal with the rest automatically according to the dynamically updated and synchronised service conditions: the last service operation would be initiated only if all the previous ones have reached the positive completion states one after another.

In Figure 7.21, an example operation chain is seen reasoned from input of an EC2 instance subject and an EC2 instance type parameter. There are three operations in the chain: “top VM”, “resize VM” and “start VM”. For the selected service instance, the reasons why the three operations can be composed into a sequenced chain rest on the following facts. Firstly, the real-time condition status of the subject decides the first operation. In the example, the state of the EC2 instance is “running”, which complies with the precondition requirement. Secondly, there are post-condition and precondition matches from first operation to another, and so are the subsequent operations one another. Seen from “Pre condition” and “Expected condition change” columns in Figure 7.21, there are exact pair between the three operations (from “EC2 Instance Status=running” to “EC2 Instance Status=stopped”, and from “EC2 Instance Status=stopped” to “EC2 Instance Status=running”). Lastly, the parameters requirement (if any) of the presented operations must all be satisfied. In this case, there are only one parameter requirement for the “resize VM” command, which can be found as the “m1.large” entered previously.

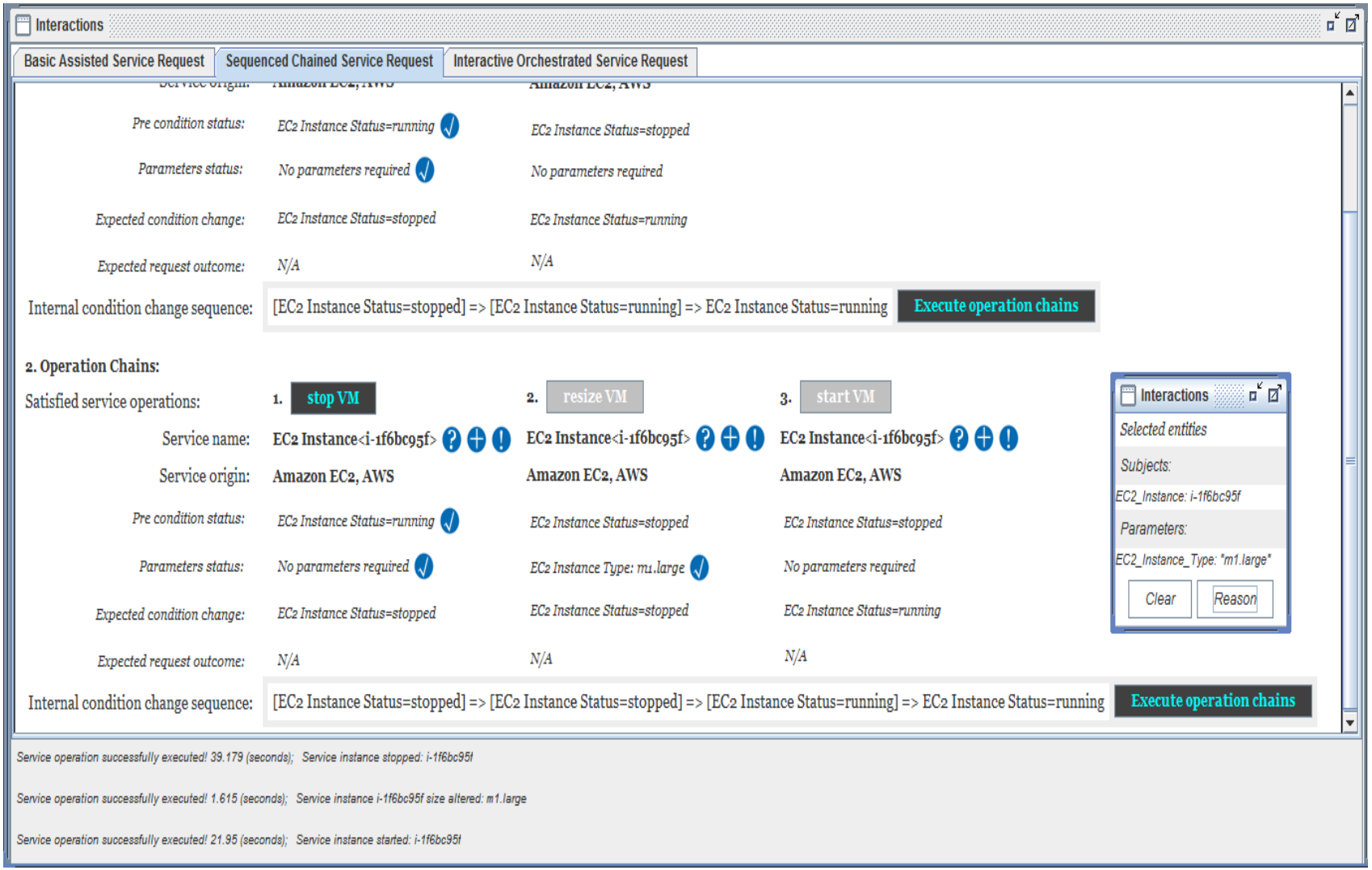


Figure 7.21 SCSR Reasoning Assistance Example

Then, as the use initiates the chained SMR, respected API requests are sent to the cloud one after another. The outcome updates of the execution can be found in the system messages at the bottom (see bottom of Figure 7.21. The whole process is also demonstrated via the native web page screenshots in Appendix C.ii.). The subsequent operations would wait until current ones finish their execution cycles. Consequently, the automatically deployed chained operations save overall execution time and efforts due to the minimum gaps and human action incurred.

7.2.4.4 Cloud Service Operation Assistance - IOSR

IOSR has a major difference in contrast with the previous reasoned operation assistances, known as the automated service operation planning and execution. This means that it would prepare the necessary service conditions and collect service parameters dynamically for the listed orchestrated service operations. When services or instances from different providers are selected for reasoning, the system seeks for possible service interactions based on whether they had any aspects in common. In Figure 7.22, an example is demonstrated with two selected cloud services: AWS EC2 and Rackspace Cloud Load Balancers.

In this scenario, the interaction entity resulted is the IP address. This is due to the fact that an IP is obviously a common entity that is recognisable for the two services selected. Hence, the likely interactions, i.e. the service operations reasoned would be centred on the IP entity of the services. Services. More specifically, by using the public IP address obtained from EC2 instances, instances of EC2 can be inserted to Rackspace Cloud Load Balancer instances in the form of load balancer nodes through operations such “add node” or “update node”.

Figure 7.22 IOSR Reasoning Assistance Example

Figure 7.23 IOSR Reasoning Assistance Example

In the meantime, as there are no instances nominated presently, the system would then allow users to either select from the instances owned or create new ones (see Figure 7.23). Then, depending on the real-time status of the instances selected, the chained interactions are implemented automatically. As seen from the log entries at the bottom of Figure 7.23, before execution, the target EC2 instance chosen was at “stopped” state and hence had no active IP; then, the system performs the “start VM” action on the instance; subsequently, as the instance becomes online and owns an IP, the “add note” request is called by using the IP address obtained dynamically; finally, as the new node presents in the load balancer node, the interaction process is completed (the whole process is also demonstrated via the native web page screenshots in Appendix C.iii).

7.2.5 Performance of Service Access and Manipulation

As a cloud service management tool that works with diverse real-world CSPs, CSRMP prototype provides an interface of service access and manipulation through a unified portal. Relying on CSAMO, it interprets the complexity which lays behind various service operation executions by revealing a diversity of operation details in a formal systematic way. Hence, this allows users to effectively view, create and amend a wide range of cloud service information via a simple structured interface. Moreover, with the series of service operation reasoning and assistances, certain service operations can be composed into groups and then executed automatically according to the dynamic status of target cloud services, where potential cloud/service interoperability issues can be eliminated accordingly.

In order to test the efficiency of the proposed approach in terms of service access and manipulation, a series of experiments are conducted over a number of cloud services. Especially, considering the diversity of service and operation types, a variety of services and several operations are selected while running the experiments. In addition, to deal with the potential deviation involved in the test data (e.g. unexpected/sight QoS differences during the tests), the results

Table 7.13 Comparison of Single SIR Access Time (Via Standard Web Portal/USAMS)

Service provider	Typical SIR / Access method	List owned cloud VM instances (IaaS)	List owned cloud database instances (PaaS)	List owned cloud files (SaaS)	List owned cloud load balancers (SaaS)	**Success rate (based on 100 tests)**
AWS	Via Web portal	< 1 sec	< 1 sec	< 1 sec	< 1 sec	>= 98%
	Via Prototype*	1.185 sec	1.032sec	1.143 sec	1.263 sec	100%
Rackspace	Via Web portal	< 2-5 sec	< 2-5 sec	< 2-5 sec	< 2-5 sec	>= 99%
	Via Prototype*	5.534 sec	5.281 sec	5.129 sec	5.483 sec	100%

are seen the average values regulated based on two factors: the tests are conducted at different time slots and on different days; the test results are obtained from several sample tests, where the minimum and maximum values are eliminated. In this way, the finalised result data is able to present their typical execution performance. Accordingly, the experiments should enable comprehensive evaluation. The sections below discuss the evaluation data in respect of both single and multiple service operation, where the comparisons between standard web portal and the proposed interface are demonstrated in details.

7.2.5.1 Performance of Single Service Operation

Evaluation of single service operation execution performance is based on 100s of tests for each sample service operation. The cloud services involved in the experiments are known as EC2, Relational Database Service (RDS), Elastic Load Balancer, Cloud Servers, Cloud Databases and Cloud Load Balancers, which belong to AWS and Rackspace respectively. For SIR response evaluation, a diversity of service instance retrieval operations are tested so as to justify the individual performances for IaaS, PaaS and SaaS services respectively. Furthermore, for SMR execution evaluation, various service instance manipulation operations are also tested, including creation, deletion, updating, etc.

A. SIR response time comparison

While attempting to retrieve these service instances, it shows many differences between accessing from standard web portal and via the prototype.

Demonstrated in Table 7.13, while accessing AWS service instance details, the official web portal offers almost instant response (less than 1 second) for all the service operations, though there is a chance for failures (success rate of 98+%). In contrast, the prototype achieves an excellent success rate, yet provides a slower access. In fact, the approximate one minute delay for each operation is caused by two factors: the API libraries (AWS Java SDK version 1.8.3) used decide the main execution time; prototype system also spends extra preparing time for processing the obtained information and other additional service operations.

On the other hand, Rackspace official web portal also offers quicker access for retrieving the service instance information. Although the time consumed for these operations are less than those for AWS, it enables almost identical success rate (99+%). Considering the performance of the prototype, despite the exceptional success rate, the information access times for Rackspace are relatively slow, seen as around 5.5 seconds in average. The reason for such delay is the dynamic synchronisation process raised by the third-party jclouds Rackspace API libraries (version 1.7.0), as currently no official API package is available.

B. SMR execution time comparison

The SMR operations involved are seen as IaaS service instance creation/termination and SaaS service instance creation/modification tasks. More specifically, the IaaS VM creation operations are deployed with plain Linux Red Hat 7.0 image on m3.large (2vCPU/7.5GB RAM) for EC2 and 4GB standard instance (2vCPU/4GB RAM) for Rackspace Cloud Servers. Then, the instances created are used for the termination tasks. On the other hand, the SaaS cloud load balancer creation and update operations are performed using a http load balancer with node adding modification tasks.

Shown in Table 7.14, overall, the success rates of all executions remain to be 100%. Note that the results are shown using “>”/”<” instead of acute figures.

This is due to some uncontrollable facts found after considerable experiments: many SMR operations often fail to execute with constant identical/typical elapsed time; instead, certain delays can always be recorded regardless of the services or the operations deployed.

Table 7.14 Comparison of Single SMR Access Time (Via Standard Web Portal/USAMS)

Service providers	Typical SMR Execution method	Create cloud VM instance (IaaS)	Terminate cloud VM instance (IaaS)	Create cloud load balancer (SaaS)	Update cloud load balancer (SaaS)	**Success rate (based on 100 tests)**
AWS	Via Web portal	> 208 sec	> 57 sec	< 1 sec	< 1 sec	100%
	Via Prototype	>= 152 sec	>= 29 sec	< 1 sec	< 1 sec	100%
Rackspace	Via Web portal	> 392 sec	> 19 sec	> 16 sec	> 7 sec	100%
	Via Prototype	>= 390 sec	>= 20 sec	>= 10 sec	>= 3 sec	100%

Specifically, for service manipulation implemented in AWS, IaaS operation tasks are completed faster through the prototype versus the official web portal, whereas the SaaS load balancer manipulation tasks execute instantly without noticeable differences regardless of the method of execution. Considering the SMRs run in Rackspace, it is found that the IaaS tasks are completed with the same resulted for both execution methods. This is due to the similar dynamic progress update and synchronisation utilised for both the web portal and the API call; nevertheless, the load balancer instance manipulation tasks tend to consume more time while launching from the standard web portal.

7.2.5.2 Performance of Multiple Service Operations

One of the benefits provided by the prototype is that it allows users to combine a series of service operations and execute them in a certain preferred manner. Based on the previous data recorded from single service operation experiments, it is expected that it should enable effective and efficient service manipulations due to fewer expected execution steps incurred as well as less overall execution time required, in contrast with ordinary web portal-based multiple tasks deployment.

Although the prototype supports initiating multiple service operations across different cloud service providers, it is difficult for the task to be implemented via different web portals for the distinct services involved, especially for concurrent

operations. Hence, the experiments are conducted within a single real-world cloud. For comprehensive evaluation of the series of proposed multiple service operations execution, both concurrent and chained SMR operation execution performances are tested, using AWS EC2 platform. The reason for choosing the IaaS platform is twofold: some SMR options of the service can be composed into operation chains one another; after execution, the SMR operations have high success rates, whereas the elapsed time is neither too short nor too long.

A. Concurrent service operations

While concurrent service operations may be initiated for both SIR and SMR, two EC2 service operations are selected to test the performance differentiations of the proposed approach versus standard task implementation.

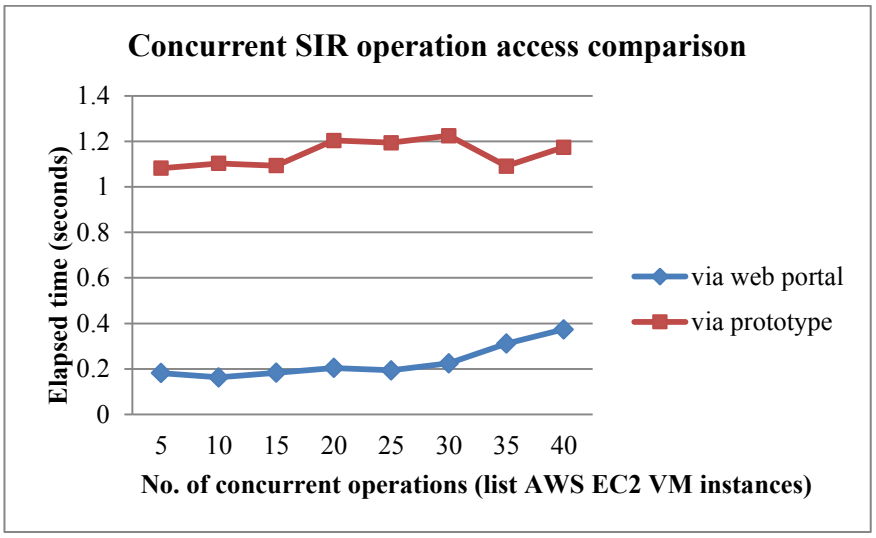


Figure 7.24 Comparison of Multiple SIR Operations Execution

As seen in Figure 7.24, for SIR such as “list all EC2 instance” (information), distinct varying patterns are found depending on the total number of instances (operations) involved. More specifically, for web portal SIR tasks, despite completing rather quickly, the responses tend to take longer while the number of the instances grows. In contrast, the trend does not apply the proposed prototype. Instead, the access time varies slightly around 1.2 seconds

regardless of the number of instances it processes. Due to the account limit (40 instances per user), it is not possible to test with more instances and demonstrate further trends.

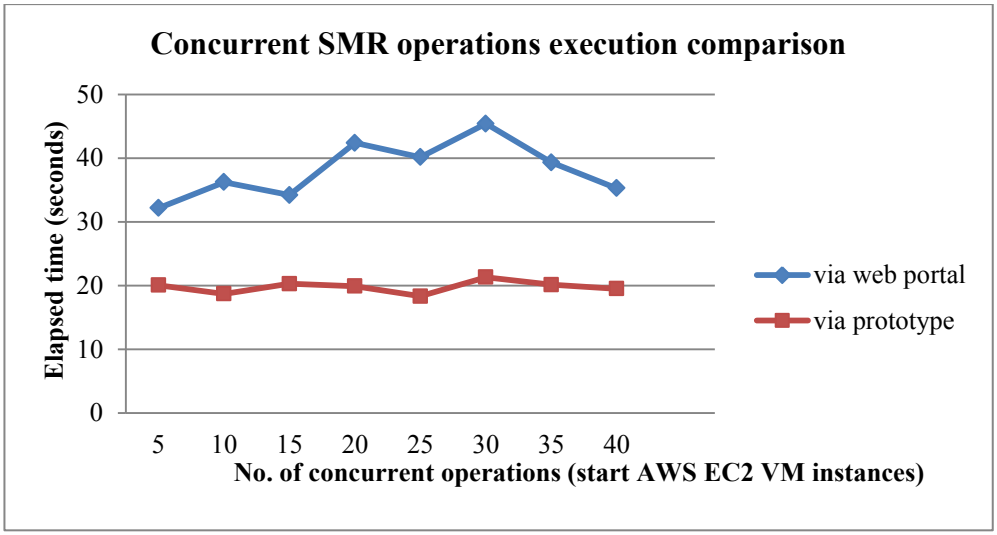


Figure 7.25 Comparison of Multiple SMR Operations Execution

On the other hand, using "start instance" as the sample SMR, the command execution response times are illustrated in Figure 7.25. For the 5 to 40 tasks deployed via standard web portal, it is found that the job completion times varies between 30 and 45 seconds; for operations implemented through the prototype, the completion times appear to be identical at 20 seconds, regardless of the number of instances (tasks) involved.

B. Chained service operations

In a single cloud service environment, it is very unlikely that a user would initiate a series of chained SIRs to obtain service information, since the tasks can always be done simultaneously. Hence, the chained service operation evaluation only justifies the overall performance of sequenced SMRs. Here, the sequence of the sample chained service operations involves start, stop and resize a VM instance, which can be executed in an infinite loop (if no error occurs).

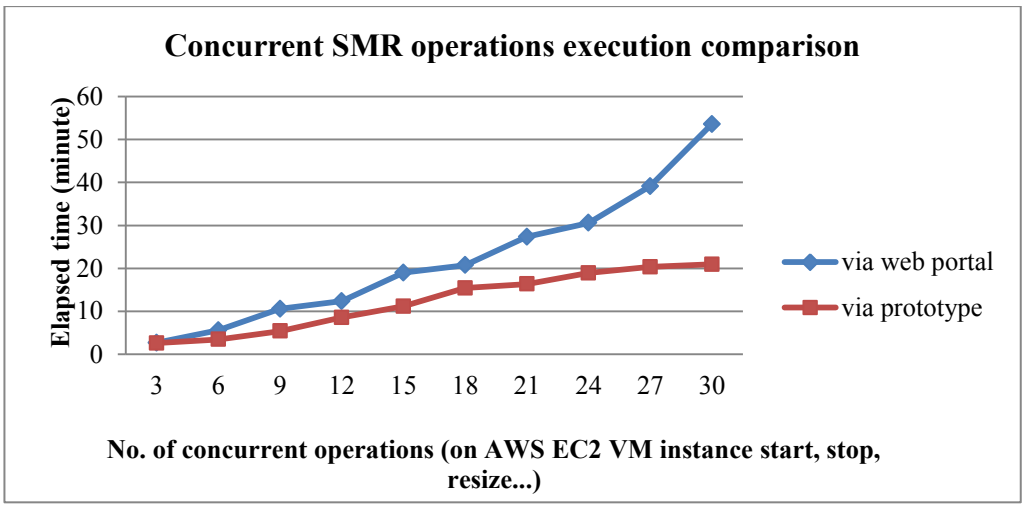


Figure 7.26 Comparison of Chained SMR Operations Execution

Figure 7.26 below demonstrates the chained tasks completion times by two different execution methods, i.e. via web portal and via prototype. Basically, as the number of the service operations increase, both methods consume more time as expected. Nonetheless, it can be seen that the increase of the completion time by using the prototype is much more gradual than which for via the standard web interface. More specifically, while executing 3 to 18 operations, the completion times between the two methods are relatively small (< 5 seconds). Yet, as more tasks are followed into the chain, the gap of them grows quickly: with some chained tasks that involve 30 service operations, the prototype can manage to complete less than half of the time required by the web portal.

7.2.6 Evaluation and Discussion

The above experiments have comprehensively tested the performance of the prototype while handling all typical types of service operations including single SIR, single SMR, multiple concurrent SIRs, multiple concurrent SMRs, and multiple chained SMRs. The analysed experiment results illustrate significant performance differences between the proposed approach and the standard web portals. Specifically, for single service operation tasks, the prototype demonstrates solid success rate while executing a diversity of operation commands, whereas there is small chances of failures while using the web portals. Although single SIR executions may take a little longer (usually 1 seconds) than the ordinary web interface, SMR operations can complete much faster (1/3 less time needed) with the prototype. Meantime, considering the multiple service operation executions, the prototype also demonstrates a better performance in overall. Although it shows that the concurrent SIR operation response times are still relatively slow while using the prototype, there is not much differences if more operations are involved; on the other hand, accessing via web portal tends to consume more and more time as the number of operations increases. On the other hand, simultaneous SMR operations can be executed much quicker in the prototype whilst the execution times are fairly stable; as a contrast, the web portal executions typically consume twice the times whilst the completion times varies significantly. Subsequently, for chained service operations, the series of sequenced SMRs can be completed sooner for prototype implementation methods. Especially, the more the chained operations are involved, the better the performance the prototype can achieve.

As illustrated in the EC2 case study, SAMOS framework can adequately model a wide range of operations. Its classifications of cloud service entities and operations enable structured specification presentation layout. The relevant operation element specifications reveal sufficient details for operation executions. As implemented in a wider service domain and across multiple CSPs, these would drive cloud service interoperability and composition (a further PaaS case example is illustrated in Appendix D). Further, to evaluate

SAMOS against other well-established cloud (service) specification frameworks/models, it provides the data comparison with OCCI, TOSCA and mOSAIC. Shown in Table 7.15, the four approaches involve dissimilar core/base model concepts with different specification semantics. They adopt distinct management tools/APIs as cloud service interfaces and enable service orchestration with own solutions. In contrast, SAMOS achieves a distinguished outcome for service management and orchestration tasks due to the flexible choices of API libraries and the lightweight operation reasoning assistances.

Meanwhile, the performance evaluation with USAMS involve covers a wide range of typical service operations. Obtained experiment results illustrate significant performance differences between the proposed approach and the standard web portals. Specifically, for single service operation tasks, the prototype demonstrates solid success rate regardless of the type/nature of operations; there is a small chance of failure while using the web portals. Although USAMS may consume a little more time (approximately 1 second) while handling single SIR operations, it facilitates SMR operations more efficiently (1/3 less time needed). Additionally, considering multiple service

Table 7.15 Comparison of Cloud Service Specification Frameworks

Approach	**Syntax/ Semantics**	**Model Core/Base Concepts**	**Management Interface**	**Service Orchestration**
OCCI	OCCI Grammar	Category, Kind, Mixin, Resource Instantiation, Collections, Discovery /Entity, Resource, Link, Action) [108]	Testing tool, doyouspeakOCCI, OCCI API	OCCI client
TOSCA	YAML	Topology Templates, Plans /Service, Node, Relationship, Requirement, Capability, Artifact, Policy, Cloud Service Archive [148]	OpenTOSCA, jclouds and PyTosca API	Pre-defined Plans
mOSAIC	OWL	Environment, Infrastructure, Resource, Runtime Component, Stateful Component, Stateless Component/etc. [103, 104]	mOSAIC API	mOSAIC Cloud Agency
SAMOS	OWL	Entity and operation classifications, Entity data type specifications, Entity operational relationship Specifications /etc.	USAMS prototype tool, flexible choice of API libraries via OCSO API	Lightweight automatic reasoning

operations, USAMS demonstrates a better overall performance. For concurrent SIRs, despite the slower responses for a small number of operations, there is no perceptible time increase despite more tasks involved. In contrast, accessing via web portal tends to consume increasingly more time as the number of operations arises. On the other hand, simultaneous SMR operations can be executed much more efficient through USAMS whilst the execution times appear to be stable. As a contrast, the web portal executions typically consume twice of the times whilst the completion times varies significantly. These results suggest the proposed approach a competent solution to enable effective and efficient cloud service operations.

7.3 Summary

This chapter has demonstrated a series of real-world cloud service case studies to validate the modelled service specifications and the enabled cloud service assistance functions. Considering the range of service recommendation, retrieval and evaluation functions, the proposed AoFeCSO is capable of comprehensively describing the wide range of cloud service features, characteristics and properties. Utilising such as the knowledge source, the CSR sub system can display comprehensive service descriptions and evaluations and enable effective service search, recommendation and comparison tasks. On the other hand, the SAMOS approach is able to model the granular aspects of cloud service operations regardless of the service provides or types. As the CSAMO and USAMS sub system are deployed based on the approach, they can provide a unified interface for efficient cloud service remote management and orchestration tasks. Accordingly, these validate the proposed ontologies and approaches with solid experiments and evaluations.

Chapter 8 Conclusions

The research undertaken for this thesis has enabled the development of a semantic-driven framework that integrates a series of the proposed approaches, including ontology modelling extensions, a service operation modelling approach, two new cloud service semantic models and a prototype tool. Together, they serve to provide the versatile cloud service recommendation and management assistances for different types of users. These research outcomes involve both the traditional and latest theory support, and are backed by the latest service modelling and manipulation technologies (e.g. PLN, OWL2, open cloud APIs).

This chapter discusses the above research outcomes in terms of how well they achieve the research objectives defined previously and fulfil the different individual requirements involved. Next, the conclusions are reached and the contributions are presented. Finally, the future research directions are outlined.

8.1 Critical Analysis

8.1.1 Objective I: Agility-oriented Cloud Service Modelling with OWL2 Natively-supported Fuzzy Extensions for Collaborative Service Search, Recommendation and Retrieval

The first thesis objective is to develop an approach to effectively assist cloud service search, recommendation and retrieval tasks. The objective has been will accomplished by the successful delivery of the following requirement via the AoFeCSO along with CSR (prototype) sub system.

R1: Scale of Cloud Service Modelling

Recently, despite many cloud service ontologies being presented, they can seldom model services of different models and functions. In contrast, the proposed cloud service modelling approach benefits from a loosely-coupled ontology foundation design, known as the flexible membership classifications

and maximum deployment of ontology property specifications. These consequently enable to maintain knowledge of diverse cloud service concepts and aspects from distinct abstraction levels and service delivery models within a single information source (refer to section 4.1.1). The requirement for the full modelling scale is therefore fulfilled.

R2: Granularity of Cloud Service Modelling

While covering cloud service concept and property specifications, the existing work often outlines the high level aspects only, without any specific details. This results into difficulties while understanding, comparing and evaluating cloud services with similar specifications. The requirement for granular service specifications is addressed by a series of ontology construction techniques including: in-depth cloud service object property assertion, categorised data property assertion and multi-sourced annotation property assertion (refer to section 4.1.3). Together, they adequately specify the fundamental details of a wide range of relevant aspects and concepts for each base cloud entity modelled. The requirement for the high modelling granularity is therefore fulfilled.

R3: Modelling Interactive Cloud Entities

In fact, there are various forms of interactions among many cloud service entities, e.g. connections between service function, features, properties, characteristics, and even providers. Yet, these relationships are often ignored or poorly disclosed in the existing ontologies. In this thesis, owning to the adoption of ReasoningOP ontology design pattern, the requirement for modelling interactive cloud entities is fulfilled. Specifically, this is achieved by explicit cloud service and concept relationship assertions (refer to section 4.1.3).

R4: Preciseness of Service Specification:

Indeed, cloud service specifications usually incur vague terms and descriptions. Fundamentally, these are due to the agile and adaptable nature of cloud services and resource provisions. Historically, although ontology techniques has

been widely used to provide quality semantics for service modelling tasks, the conventional DL consistency restricts the modelling preciseness whiling dealing with uncertainties. As a solution towards the specification precision requirement, a fuzzy extension framework is proposed. It provides a series of fuzzy scenarios to deal with different fuzziness specification and control needs, by using OWL2 natively supported assertion applications with the latest syntax features (refer to section 4.2.1 and 4.2.2).

R5: Scalability, Evolvability and Maintainability of the Cloud Service Ontology

Benefitting from the loosely-coupled ontology modelling foundation, the proposed AoFeCSO owns high scalability that accepts any forms of new information or updates. In the meantime, the adoption of the ReasoningOP design pattern guarantees the logic consistency of all the information specified and presented. It allows knowledge inference where new knowledge may be reasoned whenever the ontology is changed and updated. This model evolution process can be managed automatically by an ontology specification management mechanism (refer to section 4.2.3). Further, the maintainability of AoFeCSO is enhanced since users are allowed to input knowledge where applicable. This collaborative manner of ontology maintenance would significantly enhance the resourcefulness and creditability of the ontology.

R6: Knowledge Usage and Application of the Cloud Service Ontology

With the modelled cloud service knowledge and specifications, a wide range of assistances can be enabled. For service search tasks, AoFeCSO can facilitate various search activities regardless of using keywords or filters. For service recommendation tasks, it can provide weighted recommendations according to the individual user profile preferences. For additional service evaluation and comparison tasks, it allows to analyse and formulate service agility profiles so as to distinguish services even if they own many similarities (refer to section 6.2,

more details to be found in section 8.1.3). The requirement is fulfilled accordingly.

8.1.2 Objective II: Cloud Service Access and Manipulation Operation Modelling and Unified Service Management Portal

The second thesis objective is to enable unified cloud service access, manipulation and dynamic orchestration. The objective has been well accomplished by the successful delivery of the SAMOS approach along with USAMS (prototype) sub system.

R1: Modelling Operations Across Distinct Service Delivery Models and Levels

Many solutions are proposed to drive and enhance cloud service operation tasks across distinct service delivery models and levels. Yet, most of the existing work can only deal with certain specific service categories or function types. To fill the research gaps and fulfil the requirement, this thesis involves a series specification approaches that formulates a common cloud service operation framework that can be applied to any cloud service operations regardless of the service functions/types/models/levels.

R2: Modelling Cloud Service Operation Entities from Different CSPs

For different CSPs, many entities involved in cloud service operations are heterogeneous due to the differences exist in the service standards, technologies, terms, etc. This brings difficulties in entity specifications whilst it incurs interoperability issues for operation implementation. Targeting these issues, a novel cloud service operation specification approach is proposed. It can adequately model such complexity via cloud service entity and operation classification and cloud service entity data type specification (refer to section 5.1.1 and 5.1.2). Subsequently, this approach provides an effective solution that fulfils the requirement.

R3: Service Composition Enhancement

Indeed, many cloud service entities can act interactively for certain composited operation tasks, either within a single large scale cloud or across multiple clouds. The requirement for service composition enablement is fulfilled by effectively modelling such interactive relationships for the relevant service entities universally. It involves declaration of diverse service entity operational relationships, plus the detailed specification of operation pre condition, parameter, outcome, post condition, etc. (refer to section 5.1.3). As a result, these specifications can provide adequate information to assist service composition during tasks preparation and execution.

R4: Unified Cloud Service Operation Interface

Presently, CSCs often need to use different management portals for operations implemented over different CSPs. Towards the requirement of enabling a common interface for comprehensive management tasks, the thesis provides the design of a unified cloud service management interface. With its structured and interlinked cloud service operation presentation and control panels, the interface allows CSCs to access, navigate and manipulate cloud services/resources over multiple clouds (refer to section 6.3, more details to be found in section 8.1.3).

R5: Service Operation Reasoning Assistances

An additional requirement of cloud service operation assistance is handled by the proposed service operation reasoning assistance applications (refer to section 5.3). Considering basic assistances such as entity and condition preparation and verification for single operation, BASR is developed. For ease of concurrent service operation tasks, CCSR is proposed. To automatically execute a series of service operations with an appropriate schedule, SCSR is designed. Finally, for complicated combined service orchestration tasks, IOSR serves to dynamically prepare the operations and manage the executions.

Accordingly, these ought to fulfil the possible needs for diverse operation assistance requirements.

8.1.3 Objective III: Validation with Approach Integration and Prototype Tool Implementation

The third objective of the thesis is to implement a prototype tool to utilise the integrated ontology knowledge for versatile cloud service assistance tasks. On the one hand, AoFeCSO along with the CSR (prototype) sub system serves to provide relevant cloud service search, recommendation, retrieval, and evaluation assistance. On the other hand, CSAMO along with the USAMS (prototype) sub system serves to enable a unified cloud service management portal for service access, manipulation and orchestration tasks. Table 8.1 and 8.2 summarises the functions achieved on utilisation of the modelled cloud service specifications.

Shown in see Table 8.1, considering the search and recommendation relevant functions, a large variety of cloud service concepts, concept aspects and properties are widely processed in CSR (refer to section 6.2). More specifically, cloud service functions (e.g. compute, storage), features (e.g. protocol/API support) and characteristics (e.g. scalability, agility) aspects can be used for all sorts of search/recommendation/retrieval/evaluation relevant tasks. Other specifications, including service delivery and deployment models, parties and roles, other properties such as SLA and reliability, can also participate in service search, recommendation and comparison tasks as need. As discussed earlier in section 7.1.4, these provide the feasibility of comprehensive cloud service profile analysis and data evaluation during search and recommendation processes. Consequently, this greatly extends the current practices by retrieving more accurate service candidates with more flexible search and recommendation controls.

Table 8.1 Cloud Service Specifications Toward Service Recommendation Relevant Functions

Function / Service aspects	Service description (annotation)	Service search	Service recommendation	Service comparison	Service evaluation
Service functions	√	√	√	√	√
Service features	√	√	√	√	√
Service characteristics	√	√	√	√	√
Service delivery models	√	√	√	√	
Service deployment models	√	√	√	√	
Service party/Roles	√	√	√	√	
Other service properties	√	√	√	√	

Table 8.2 Cloud Service Operation Specifications Toward Service Management Relevant Functions

Function / Service operation aspects	Element description (annotation)	Requirement/ Element dynamic lookup	Requirement/ Element dynamic fulfilment	Requirement/ Element dynamic verification
Operation Classification	√			
Operation PreCondition	√	√	√	√
Operation PostCondition	√	√	√	√
Operation Subject	√	√	√	√
Operation Parameter	√	√	√	√
Operation Outcome	√	√	√	√
Operation Orchestration		√	√	√

Furthermore, as Table 8.2 shows, SAMOS framework provides compressive cloud service operation specifications, which enable a diversity of service operation management and assistance functions in USAMS (prototype) sub system (refer to section 6.3). Specifically, CSAMO offers description and reasoning support for diverse service operations and relevant elements involved.

These specifications, including operation classifications, parameters, outcome, pre/post conditions, etc. provide fundamental information that guides operation execution process. The prototype is, therefore, able to facilitate various cloud service operation tasks by satisfying advanced needs such as dynamic operation requirement lookup, fulfilment, verification and orchestration. As compared with other solutions in section 7.2.6, this outperforms alternative solutions by enabling not only performance and reliability, but also a range of assistance functions toward better service management operations.

8.1.4 Objective IV: Evaluation with Real-world Cloud Service Case Studies

For critical evaluation requirements, the thesis involves several real-life case studies and experiments using popular real-world cloud services from multiple clouds and service delivery models. Firstly, quantitative literature and more than 100 companies are researched and investigated to construct AoFeCSO. Considering the cloud service search, recommendation enhancement studies, specifications of over 200 cloud services are processed for the search and recommendation tasks (refer to section 7.1.1 and 7.1.2). Secondly, for specific cloud service specification retrieval and evaluation study, Google AppEngine was selected as the typical example and qualitatively examined (refer to section 7.1.3). Thirdly, the specification processing performance experiment and formal ontology evaluation are performed. Consequently, it shows that the proposed approach can achieve effective cloud service specification towards a combination of service search, recommendation and evaluation requirement.

In the meantime, to evaluate the comprehensiveness of the SAMOS modelling framework, two series of operations from IaaS and SaaS model are studied explicitly (refer to section 7.2.1 and 7.2.2). Examples are demonstrated with regard to the enablement of the unified cloud service management interface (refer to section 7.2.3). Next, to assess the proposed operation reasoning assistance applications, a series of operation task examples are tested (refer to section 7.2.4). Subsequently, to examine the service operation execution

performance of the prototype, extensive experiments are conducted on several cloud services of distinct functions, delivery models and CSPs (refer to section 7.2.5). These suggest that the proposed approach enables generic cloud service operation modelling and can facilitate effective service operations via the common management interface.

8.2 Conclusions and Contributions

The continuously propagated cloud service has imposed strong requirements for comprehensive cloud service specification models as well as effective service recommendation systems. Meanwhile, existing cloud (service) models cannot cover comprehensive and in-depth service specifications in regard of diverse concepts and their interactions across different function categories and abstraction levels, whereas current service recommendation tools fail to handle the detailed aspects of the various and unique cloud service characteristics, properties and orchestrations. In addition, none of the current practices attempts to capture and deal with the fuzzy specification and facts that are widely and frequently encountered; this consequently prevents existing models and service assistance tools from facilitating versatile service search, retrieval and recommendation tasks.

The thesis aims towards a cloud service semantic specification approach which takes into consideration of the combination of service function, feature, delivery model, operation, orchestration, etc. concepts and aspects so as to enable versatile service search, recommendation, retrieval, and management assistances. The following work has been undertaken during the study.

8.2.1 Contribution I: OWL2 Natively Supported Fuzzy Extensions

The thesis demonstrated an OWL2 fuzzy extension framework that can deal with a wide range of specification fuzziness. The extension benefits from OWL2 native syntax application for maximum compatibility. For effective fuzziness

representation, it involves both data-oriented fuzzy weight assertion and additional fuzzy rating details annotation assertion.

Moreover, unlike most existing approaches which require additional fuzziness interpretation and reasoning mechanism, the fuzzy extended ontology can be easily interpreted and reasoned by ordinary classic ontology tools and reasoners. Indeed, the approach can be widely adopted while modelling vague or uncertain specifications for other domains.

8.2.2 Contribution II: AoFeCSO

This thesis presented a novel cloud service semantic model named AoFeCSO. It owns the following four main features: 1) it introduces multiple sourced annotation assertions for trustful cloud services descriptions; 2) it employs functionally categorised DP assertions and a diversity of data types for comprehensive service data specifications; 3) it discloses in-depth service details regarding services' characteristics, features, functionalities, etc. by exploring their fundamental sub-concepts involved; 4) it reveals explicit cloud service and concept relations through both asserted and inferred axioms in the form of individual-to-class and individual-to-individual OP and property characteristics assertions.

Additionally, different from other models which are managed exclusively and deployed statically, AoFeCSO is maintained collaboratively and can evolve autonomously, and hence remains active. Indeed, the proposed collaborative cloud service rating mechanism enhances the presentation of several cloud service specifications. Hence, the overall building source of the ontology becomes much wider and more accurate, whereas these continuously imported dynamic aspects actively drive AoFeCSO to evolve progressively.

8.2.3 Contribution III: SAMOS framework

This thesis proposed a cloud service operation specification approach which can be applied to diverse cloud service delivery models and resource types,

namely SAMOS. The modelling framework can reveal comprehensive information with regard to the involved service entities, their attributes and relationships, plus a series of operational elements including parameters, conditions and outcomes.

Further, owning to its ontological modelling techniques, the approach also enables a series of service operation reasoning assistances. They can provide intelligent and automated solutions for advanced multi-provider operation tasks such as simultaneous, chained and service orchestration actions.

8.2.4 Contribution IV: CSAMO

Based on SAMOS framework, CSAMO was implemented. It incorporates numerous cloud service operation specifications from popular cloud vendors such as Amazon, Rackspace. It demonstrated granular operation descriptions for each granular cloud service operations from three hierarchical initiation levels, known as cloud service level, CSI level and PSSA level.

The presented cloud service operation specifications can be widely utilised, such as to serve as a comprehensive knowledge source for operation annotations, to compare or evaluate operations for similar services. Additionally, along with the proposed service API mapping mechanism, they can enable efficient service remote management tasks towards customisability requirement.

8.2.5 Contribution V: CSRMP prototype tool

To validate and evaluate the effectiveness and comprehensiveness of the proposed cloud service ontologies and modelling approaches, a joint prototype tool was developed to facilitate a combination of cloud service search, recommendation, retrieval, comparison, evaluation, access, manipulation, and orchestration tasks.

On the one hand, CSR sub system provides an effective solution for cloud services search, recommendation, retrieval and evaluation from distinct service

categories and delivery models the performance and effectiveness evaluation results suggest that it is a promising means to overcome various existing limitations. On the other hand, USAMS sub system enables a unified cloud service access and manipulation via a structured management interface. This is validated through considerable experiments that are conducted over Amazon and Rackspace IaaS, PaaS and SaaS clouds. The test results suggest that USAMS can provide competitive service operation effectiveness and efficiency, especially while handling groups of operation tasks.

8.3 Future Work

Considering future research directions on cloud service search, recommendation and comparison enhancement, the future work will target at extending the proposed framework and tool for extended ontology modification and evolution, e.g. to allow CSPs to add services, change service specifications, etc.; to allow CSBs to specify service interactions and orchestrations, etc.; to allow CSCs to complete service usability ratings, reviews, etc. It is believed that this collaborative manner of cloud service ontology specification, maintenance and update to be a distinguished means in providing knowledge sources for service search, retrieval and recommendation tasks.

In the meantime, for future development on cloud service remote management tasks, the existing work will be extended by introducing the service recommendation engine and the service interaction agent. The recommendation module should enable more user friendly service selection and operation experiences. The service interaction agent would drive more effective service compositions with enhanced operation reasoning applications.

References

1. Amazon EC2 User Guide, http://awsdocs.s3.amazonaws.com/EC2/latest/ec2-ug.pdf. Accessed 12 Aug 2015
2. Amazon S3 Developer Guide, http://awsdocs.s3.amazonaws.com/S3/latest/s3-dg.pdf. Accessed 12 Aug 2015
3. Antwerp ALV, Scoboria K and Santos JR, “Security Guidance for Critical Areas of Focus in Cloud Computing v3.0”. https://cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf. Accessed 12 Aug 2015
4. Apache Jclouds, http://jclouds.apache.org/. Accessed 30 Jul 2015
5. Apache Libcloud, https://libcloud.apache.org/. Accessed 12 Aug 2015
6. Arthi T and Hamead HS, “Energy Aware Cloud Service Provisioning Approach for Green Computing Environment”, *International Conference on Energy Efficient Technologies for Sustainability (ICEETS),* pp. 139-144, 2013
7. Aversa R, Martino BD, Moscato F, Petcu D, Rak M and Venticinque S, “An Ontology for the Cloud in mOSAIC”, *Cloud Computing Book: Cloud Computing: Methodology, System, and Applications, CRC Press*, pp. 467-486, 2011
8. Bastião Silva LA, Costa C and Oliveira JL, “A Common API for Delivering Services over Multi-vendor Cloud Resources”, *Journal of Systems and Software*, vol. 86, no. 9, pp. 2309-2317, 2013
9. Battle S, Bernstein A, Boley H, Grosof B, Gruninger M, Hull R and Kifer M, “Semantic Web Service Ontology (SWSO)”, http://www.daml.org/services/swsf/1.0/swso/. Accessed 13 Aug 2015
10. Behrendt M, Glasner B, Kopp P, Dieckmann R, Breiter G, Pappe S, Kreger H and Arsanjani A, “Introduction and Architecture Overview: IBM Cloud Computing Reference Architecture 2.0”, https://www.ibm.com/developerworks/community/files/form/anonymous/api/library/20eaa907-3440-482c-a234-65c3584bdd7c/document/e817254b-f0b8-4e6d-8980-ef753c531825/media/CCRA_2.0_NonConfidential.pdf. Accessed 13 Aug 2015
11. Behl A and Behl K, “An Analysis of Cloud Computing Security Issues”, *World Congress on Information and Communication Technologies (WICT)*, pp. 109-114, 2012

12. Beloglazov A, Abawajy J, Buyya R, “Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing”, *Future Generation Computer Systems*, vol. 28, no 5, pp 755-768, 2012
13. Bernabe JB, Marin Perez JM, Alcaraz Calero JM, Garcia Clemente FJ, Perez GM, Gomez Skarmeta AF, “Semantic-aware Multi-tenancy Authorization System for Cloud Architectures”, *Future Generation Computer Systems*, vol. 32, pp. 154-167, 2014
14. Berners-Lee T, Hendler J and Lassila O, “*The Semantic Web”, Scientific American*, pp. 29–37, 2001
15. Bibi S, Katsaros D and Bozanis P, “Business Application Acquisition: On-Premise or SaaS-Based Solutions”, *IEEE Software*, vol. 29, no. 3, pp. 86-93, 2012
16. Bobillo F and Straccia U, “Fuzzy Ontology Representation Using OWL2”, *International Journal of Approximate Reasoning*, vol. 52, no. 7, pp. 1073-1094, 2011
17. Bobillo F and Straccia U, “fuzzyDL: An Expressive Fuzzy Description Logic Reasoner", *IEEE International Conference on Fuzzy Systems*, no. 1-6 pp. 923-930, 2008
18. Bobillo F, Delgado M and Gomez-Romero J, “Delorean: A Reasoner for Fuzzy OWL2”, *Expert Systems with Applications*, vol. 39, pp. 258-272, 2012
19. Bobillo F and Straccia U, “An OWL Ontology for Fuzzy OWL2, Foundations of Intelligent Systems”, *Lecturer notes in Computer Science*, vol. 5722, pp. 151-160, 2009
20. Bondi AB, “Characteristics of Scalability and Their Impact on Performance, *Proceeding of the 2nd ACM international workshop on Software and performance, pp. 195-203, 2000*
21. Brambilla M, Ceri S, Facca FM, Celono I, Cerizza D and Valle ED, “Model-Driven Design and Development of Semantic Web Service Applications”, *ACM Transactions on Internet Technology*, vol. 8, no. 1, art. 3, 2007
22. Burton-Jones A, Storey VC, Sugumaran V and Ahluwalia P, “A Semiotic Metrics Suite for Assessing the Quality of Ontologies”, *Data & Knowledge Engineering*, vol. 55, no. 1, pp. 84-102, 2005
23. Buyya R, Vecchiola C and Thamarai S, "Master Cloud Computing: Foundations and Applications Programming”, *Elsevier*, Waltham, USA, pp. 3-27, 2013

24. Byung CT, Urgaonkar B and Sivasubramaniam A, “Cloudy with a Chance of Cost Savings”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1223-1233, 2013
25. Cáceres J, Vaquero LM, Rodero-Merino L, Polo and Hierro JJ, “Service Scalability over the Cloud”, *Handbook of Cloud Computing, Springer US*, pp. 357-377, 2010
26. Cisco WebEx Overview, http://www.webex.co.uk/why-webex/overview.html. Accessed 12 Aug 2015
27. Chapman C, Emmerich W, M´arquez FG, Clayman S and Galis A, “Elastic Service Definition in Computational Clouds”, *IEEE/IFIP Network Operations and Management Symposium Workshops*, pp. 327-334, 2010
28. Cloud Computing Use Cases group, Cloud Computing Use Cases White Paper. http://opencloudman ifesto.org/Cloud_Computing_Use_Cases_Whitepaper-4_0.pdf. Accessed 10 Oct 2014
29. Cohen B, “PaaS: New Opportunities for Cloud Application Development”, *Computer*, vol. 46, no. 9, pp. 97-100, 2013
30. Cunningham H, Maynard D, Bontcheva K, Tablan V, Ursu C, Dimitrov M, Dowman M, Aswani N, Roberts I and Li Y, “Developing Language Processing Components with GATE Version 5”. *University of Sheffield*, http://gate.ac.uk /releases/gate-5.0-build3244ALL/doc/tao/splitch1.html. Accessed 11 Aug 2015
31. d'Aquin M, Motta E, Sabou M, Angeletou S, Gridinoc L, Lopez V and Guidi D, “Toward a New Generation of Semantic Web Applications”, *IEEE Intelligent Systems*, vol. 23, no. 3, pp. 20-28, 2008
32. Das-neves F, Fox EA and Yu X, “Connecting topics in document collections with stepping stones and pathways", *Proceedings of the 14th ACM CIKM*, pp. 91- 98, 2005
33. De Chaves SA, Uriarte RB and Westphall CB, “Toward an Architecture for Monitoring Private Clouds”, *IEEE Communications Magazine*, vol. 49, no. 12, pp. 130-137, 2011
34. Deltacloud, https://deltacloud.apache.org/. Accessed 22 May 2014
35. Demchenko Y, Ngo C, de Laat C, Rodriguez J, Contreras LM, Garcia-Espin JA, Figuerola S, Landi G and Ciulli N, “Intercloud Architecture Framework for Heterogeneous Cloud based Infrastructure Services Provisioning On-Demand”. *IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp 777-784, 2013

36. Deng Y, Head RH, Kochut A, Munson J, Sailer A and Shaikh H, “Introducing Semantics to Cloud Services Catalogs”, *IEEE International Conference on Services Computing*, pp.24-31, 2011
37. Dropbox Platform developer guide, https://www.dropbox.com/developers/reference/devguide. Accessed 19 Aug 2015
38. Emeakaroha VC, Netto MAS, Calheiros RN, Brandic I, Buyya R and De Rose CAF, “Towards autonomic detection of SLA violations in Cloud infrastructures”, *Future Generation Computer Systems*, vol. 28, no. 7, pp. 1017-1029, 2011
39. Etinski M, Corbalan J, Labarta J, Valero M, “Understanding the Future of Energy-performance Trade-off via DVFS in HPC Environments”*, Journal of Parallel and Distributed Computing*, vol. 72, no 4, pp 579-590, 2012
40. Fang D, Liu X, Romdhani I, “A Loosely-coupled Semantic Model for Diverse and Comprehensive Cloud Service Search and Retrieval”, *The Fifth International Conference on Cloud Computing*, pp.6-11, 2014
41. Fang D, Liu X, Romdhani I and Claus P, “An Approach to Unified Cloud Service Access, Manipulation and Dynamic Orchestration via Semantic Cloud Service Operation Specification Framework”, *Journal of cloud computing: Advances, Systems and Applications*, vol. 4, no.14, pp. 1-20, 2015
42. Fang D, Liu X, Romdhani I and Zhao H, “Towards OWL2 Natively Supported Fuzzy Cloud Ontology”, *36th Annual Computer Software and Applications Conference Workshops (COMPSACW)*, pp. 328-333, 2012
43. Fang D, Liu X, Liu L and Yang H, “OCSO: Off-the-cloud Service Optimization for Green Efficient Service Resource Utilization”, *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 3, no. 9, pp. 1-17, 2014
44. Fang D, Liu X, Liu L, Yang H, “TARGO: Transition and Reallocation Based Green Optimization for Cloud VMs”, *IEEE International Conference on Green Computing and Communications (GreenCom)*, pp 215-223, 2013
45. Federici C, “Cloud Data Imager: A Unified Answer to Remote Acquisition of Cloud Storage Areas”, *Digital Investigation*, vol. 11, no. 1, pp. 30-42, 2014
46. Fjellheim T, Milliner S, Dumas M. and Vayssie`re J, “A Process-based Methodology for Designing Event-based Mobile Composite Applications”, *Journal of Data & Knowledge Engineering*, vol. 61, no. 1, 2007
47. Fog. http://fog.io/. Accessed 02 Aug 2015

48. Gagnon S, Nabelsi V, Passerini K and Cakici K, “The Next Web Apps Architecture: Challenges for SaaS Vendors”, *IEEE IT Professional*, vol. 13, no. 5, pp. 44-50, 2011
49. Gam E. and Reich S, “An Analysis of the Applicability of User Trails in Web Applications”, Web Engineering Workshop, pp. 89-94. 2004
50. Gangemi A, “Ontology Design Patterns for Semantic Web Content”, *Lecture Notes in Computer Science*, vol. 3729, pp. 262-276, 2005
51. Goertzel B, Iklé M, Goertzel IF and Heljakka A, “Probabilistic Logic Networks - A Comprehensive Framework for Uncertain Inference”, Springer, pp.1-148, 2008
52. GoGrid Cloud Servers Overview, https://wiki.gogrid.com/wiki/index.php/Cloud_Servers. Accessed 12 Aug 2015
53. Golbreich C, Wallace EK. and Patel-Schneider PF, “OWL2 Web Ontology Language New Features and Rationale”, *W3C Recommendation 27 October 2009*, http://www.w3.org/2009/pdf/REC-owl2-new-features-20091027.pdf. Accessed 05 Aug 2015
54. Goiri Í, Ll J, Berral, Oriol Fitó J, Julià F, Nou R, Guitart J, Gavaldà R and Torres J, “Energy-efficient and Multifaceted Resource Management for Profit-driven Virtualized Data Centers”, *Future Generation Computer Systems*, vol. 28, no. 5, pp. 718-731, 2012
55. Google AppEngine. https://developers.google.com/appengine/. Accessed 02 Aug 2015
56. Google Apps for Business, Web applications that increase productivity, http://www.google.com/Apps. Accessed 10 Aug 2015
57. Goscinski A and Brock M, “Toward Dynamic and Attribute Based Publication, Discovery and Selection for Cloud Computing”, *Future generation computer systems*, vol. 26, no. 7, pp. 947-970, 2010
58. Gracia J and Mena E (2012) Semantic Heterogeneity Issues on the Web, *IEEE Internet Computing*, vol. 16, no. 5, pp. 60-67
59. Grigori D, Corrales JC. and Bouzeghoub M, “Behavioural Matchmaking for Service Retrieval: Application to Conversation Protocols”, *Information Systems*, vol. 33, no. 7-8, pp. 681-698, 2008
60. Han T and Sim KM, “An Ontology-enhanced Cloud Service Discovery System”, International Multi-Conf. *Engineers and Computer Scientists (IMECS)*, no. 1, pp. 27-32, 2010

61. HEADS, HEADS project overview, http://heads-project.eu/objectives. Accessed 15 Jun 2015
62. Himmel MA and Grossman F, “Security on Distributed Systems: Cloud Security Versus Traditional IT”, *IBM Journal of Research and Development* , vol. 58, no. 1, pp. 1-13, 2014
63. Hoefer CN and Karagiannis G, “Taxonomy of Cloud Computing Services”, *proceedings of the 4th IEEE workshop on enabling the future service-oriented internet*, pp. 1345-1350, 2010
64. Hofmann P and Woods D, “Cloud Computing: The Limits of Public Clouds for Business Applications”, *IEEE Internet Computing*, vol. 14, no. 6, pp. 90-93, 2010
65. Horridge M, “A Practical Guide to Building OWL Ontology's Using Protégé 4 and CO-ODE Tools”, *The University of Manchester*, edition 1.3. http://owl.cs.manchester.ac.uk/tutorials/protege owltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf. Accessed 14 Aug 2015
66. Horridge M and Bechhofer S, “The OWL API: A Java API for OWL Ontologies”, *Special Issue on Semantic Web Tools and Systems, Semantic Web Journal*, vol. 2, no. 1, pp. 11-21, 2011
67. Huang C, Guan C, Chen H, Wang Y, Chang S, Li C and Weng C, “An Adaptive Resource Management Scheme in Cloud Computing”, *Engineering Applications of Artificial Intelligence*, vol. 26, no 1, pp. 382-389, 2013
68. Jansen W and Grance T, “Guidelines on Security and Privacy in Public Cloud Computing”, *Draft NIST Special Publication*, https://downloads.cloudsecurityalliance.org/initiatives/ guidance/NIST-Draft-SP-800-144_cloud-computing.pdf. Accessed 11 Aug 2015
69. Jeffery K and Neidecker-Lutz B, “The Future of Cloud Computing: Opportunities for European Cloud Computing Beyond 2010”, *Software & Service Architectures and Infrastructures, ICT, FP7, European commission, 2010*
70. Jeyarani R, Nagaveni N, Srinivasan S and Ishwarya C, “ISim: A Novel Power Aware Discrete Event Simulation Framework for Dynamic Workload Consolidation and Scheduling in Infrastructure Clouds”, *Advances in Intelligent Systems and Computing*, vol. 177, pp 375-384, 2013
71. Joint A and Baker E, “Knowing the past to Understand the Present – Issues in the Contracting for Cloud based Services”, *Computer Law & Security Review, Science Direct*, vol. 27, no. 4, pp. 407-415, 2011

72. Jung JJ, “Semantic Business Process Integration based On Ontology Alignment”, *Expert Systems with Applications*, vol. 36, no. 8, pp. 11013-11020, 2009
73. Kang J and Sim K, “Cloudle: A Multi-criteria Cloud Service Search Engine”, *IEEE Asia-Pacific Services Computing Conference (APSCC)*, pp.339-346, 2010
74. Kaufman LM, “Data Security in the World of Cloud Computing”, *IEEE Security & Privacy*, vol. 7, no. 4, pp. 61-64, 2009
75. Kavantzas N, Burdett D, Ritzinger G, Fletcher T, Lafon Y and Barreto C, “Web Service Choreography Description Language Version 1.0”, *W3C candidate recommendation 9 November 2005*, http://www.w3.org/TR/ws-cdl-10/. Accessed 26 Jul 2015
76. Ke C and Huang Z, “Self-adaptive Semantic Web Service Matching Method”, *Knowledge-Based Systems*, vol. 35, pp. 41-48, 2012
77. Keppeler J, Brune P and Gewald H, “A Description and Retrieval Model for Web Services Including Extended Semantic and Commercial Attributes”, *IEEE 8th International Symposium on Service Oriented System Engineering (SOSE)*, pp. 258-265, 2014
78. Kim D and Vouk MA, “A Survey of Common Security Vulnerabilities and Corresponding Countermeasures for SaaS”, *Globecom Workshops*, pp. 59-63, 2014
79. Kim S and Choi H, “An Ontology Model Framework for Supply of Active Situation Decision Service”*, IEEE Fourth International Conference on Computer Sciences and Convergence Information Technology*, pp. 1207-1212, 2009
80. Llora X, Acs B, Auvil LS, Capitanu B, Welge ME and Goldberg DE, “Meandre: Semantic-Driven Data-Intensive Flows in the Clouds”, *IEEE Fourth International Conference on eScience*, pp. 238-245, 2008
81. Li C and Li Y, “Optimal Resource Provisioning for Cloud Computing Environment”, *The Journal of Supercomputing*, vol. 62, no 2, pp. 989-1022, 2012
82. Li L, Liu D and Bouguettaya A, “Semantic based Aspect-oriented Programming for Context-Aware Web Service Composition”, *Information Systems*, vol. 36, no. 3, pp. 551-564, 2011

83. Li J, Li B, Wo T, Hu C, Huai J, Liu, L and Lam KP, “CyberGuarder: A Virtualization Security Assurance Architecture for Green Cloud Computing”, *Future Generation Computer Systems*, vol. 28, no. 2, pp. 379-390, 2012
84. Li Q, Wang Z, Li W, Cao Z, Du R and Luo H, “Model-based Services Convergence and Multi-Clouds Integration”, *Computers in Industry*, vol. 64, no. 7, pp. 813-832, 2013
85. Li Y, Krishnamurthy R, Vaithyanathan S. and Jagadish HV, “Getting Work Done on the Web: Supporting Transactional Queries”, *Proceedings of the 29th ACM SIGIR*, pp. 557-564, 2006
86. Lim HC, Babu S and Chase JS, “Automated Control for Elastic Storage”, *Proceedings of the 7th ACM International conference on automated computing*, pp. 1-10, 2010
87. Liu Y and Agah A, “A Prototype Process-Based Search Engine”*, IEEE International Conference on Semantic Computing*, pp.481-486
88. Liu F, Tong J, Mao J, Bohn RB, Messina JV, Badger ML and Leaf DM, “NIST Cloud Computing Reference Architecture”, http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909505. Accessed 20 Aug 2015
89. Lombardi F and Pietro RD, “Secure Virtualisation for Cloud Computing”, *Journal of Network and Computer Applications*, vol. 34, no. 4, pp.1113-1122, 2011
90. Loutas N, Peristeras V, Bouras T, Kamateri E, Zeginis D and Tarabanis K, “Towards a Reference Architecture for Semantically Interoperable Clouds”*, IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom)*, pp 143-150, 2010
91. Lukasiewicz T and Straccia U, “Managing Uncertainty and Vagueness in Description Logics for the Semantic Web”, *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, no. 4, pp. 291-308, 2008
92. Marinescu DC, “Cloud computing: Theory and Practice” , *Elsevier*, Waltham, USA, pp. 2-17, 2013
93. Marston S, Li Z, Bandyopadhyay S, Zhang J and Ghalsasi A, “Cloud Computing — The Business Perspective, Decision Support Systems”, Elsevier, vol. 51, no. 1, pp. 176-189, 2011
94. Martin D and Domingue J, “Semantic Web Services, Part 1”, *IEEE Intelligent Systems*, vol. 22, no. 5, pp. 12-17, 2007

95. Martin D and Domingue J, “Semantic Web Services, Part 2”, *IEEE Intelligent Systems*, vol. 22, no. 6, pp. 8-15, 2007
96. Manno G, Smari WW and Spalazzi L, “FCFA: A Semantic-based Federated Cloud Framework Architecture”, *International Conference on High Performance Computing and Simulation (HPCS)*, pp. 42-52, 2012
97. Maven link, Features, http://www.mavenlink.com/features. Accessed 05 Aug 2015
98. McGuuinness, D.L and Harmelen F.V, “OWL Web Ontology Language Overview”, W3C recommendation 10 Feb 2004, http://www.w3.org/TR/owl-features/, Accessed 05 Aug 2015
99. Modica GD, Petralia G and Tomarchio O, “A Business Ontology to Enable Semantic Matchmaking in Open Cloud Markets”, *8th International Conference on Semantics, Knowledge and Grids (SKG)*, pp. 96-103, 2012
100. Mokhtar SB, Preuveneers D, Georgantas N, Issarny V and Berbers Y, “EASY: Efficient semAntic Service discoverY in Pervasive Computing Environments with QoS and Context Support”, *Journal of Systems and Software*, vol. 81, no. 5, pp. 785-808, 2008
101. Moreno-Vozmediano R, Montero RS and Llorente IM, “IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures”, *IEEE Computer*, vol. 45, no.12, pp.65-72, 2012
102. Moreno-Vozmediano R, Montero RS and Llorente IM, “Key Challenges in Cloud Computing: Enabling the Future Internet of Services”. *IEEE Internet Computing*, vol. 17, no. 4, pp. 18-25, 2011
103. Moscato F, Aversa R, Martino BD and Venticinque S, “An Ontology for the Cloud in mOSAIC”, *Cloud Computing: Methodology, System, and Applications. CRC Press*, pp.467-485, 2011
104. Moscato F, Fortis F and Munteanu V, “Cloud Ontology and Cloud Resources Representations”, mOSAIC public deliverables D 1.2. http://www.mosaic-cloud.eu/index.php?option=com_chronocontact&Itemid=186. Accessed 29 Jul 2015
105. Nacer H and Aissani D, “Semantic Web Services: Standards, Applications, Challenges and Solutions”, Journal of Network and Computer Applications, vol. 44, pp. 134-151, 2014
106. Naumann S, Dick M, Kern E and Johann T, “The GREENSOFT Model: A Reference Model for Green and Sustainable Software and Its Engineering”,

Sustainable Computing: Informatics and Systems, vol. 1, no 4, pp. 294-304, 2011

107. OCCI. http://occi-wg.org/. Accessed 31 Jul 2015
108. Open Cloud Computing Interface – Core, https://www.ogf.org/documents/GFD.183.pdf. Accessed 21 Aug 2015
109. Orozco JMS, "Applied Ontology Engineering in Cloud Services, Networks, and Management Systems", *Springer Science+Business Media*, pp. 23-52, 2012
110. Owens D, "Securing Elasticity in the Cloud". *Communications of the ACM*, vol. 8, no. 5, 2010
111. Parrilli DM, "Legal Issues in Grid and Cloud Computing, Grid and Cloud Computing", *Springer Berlin Heidelberg*, pp. 97-118, 2010
112. Peoples C, Parr G, McClean S, Scotney B and Morrow P, "Performance Evaluation of Green Data Centre Management Supporting Sustainable Growth of the Internet of Things", *Simulation Modelling Practice and Theory*, vol. 34, pp. 221-242, 2013
113. Petcu D, Craciun C, Neagul M, Lazcanotegui I and Rak M, "Building an Interoperability API for Sky Computing". *International Conference on High Performance Computing and Simulation (HPCS)*, pp. 405-411, 2011
114. Policy-based automated scaling for IBM SmarCloud Application Services. http://www.ibm.com/cloud-computing/uk/en/paas.html. Accessed 23 Jul 2015
115. Portmann E, Meier A, Cudré-Mauroux P and Pedrycz W, "FORA — A Fuzzy Set Based Framework for Online Reputation Management", *Fuzzy Sets and Systems*, vol. 269, pp. 90-114, 2014
116. Povedano-Molina J, Lopez-Vega JM, Lopez-Soler JM, Corradi A and Foschini L, "DARGOS: A Highly Adaptable and Scalable Monitoring Architecture for Multi-Tenant Clouds". *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2041-2056, 2013
117. Rackspace Cloud Load Balancers Developer Guide. http://docs.rackspace.com/loadbalancers/api/v1.0/clb-devguide/clb-devguide-20140630.pdf. Accessed 19 Aug 2015
118. Rackspace Next Generation Cloud Servers Developer Guide. http://docs.rackspace.com/servers/api/v2/cs-devguide/cs-devguide-20140311.pdf. Accessed 19 Aug 2015
119. Ranjan R, "The Cloud Interoperability Challenge", *IEEE Cloud Computing*, vol. 1, no. 2, pp. 20-24, 2014

120. Rico M, Caliusco ML, Chiotti O and Galli MR, “OntoQualitas: A Framework for Ontology Quality Assessment in Information Interchanges between Heterogeneous Systems”, *Computers in Industry*, vol. 65, no. 9, pp. 1291-1300, 2014
121. Rodríguez-García MA, Valencia-García R, García-Sánchez F and Samper-Zapater JJ, “Ontology-based Annotation and Retrieval of Services in the Cloud”, *Knowledge-Based Systems*, vol. 56, pp. 15-25, 2014
122. Rodríguez-García MA, Valencia-García R, García-Sánchez F and Samper-Zapater JJ, “Creating a Semantically-Enhanced Cloud Services Environment through Ontology Evolution”, *Future Generation Computer Systems*, vol. 32, pp. 295-306, 2014
123. Roman D, Lausen H and Keller U, “Web Service Modelling Ontology, WSMO final draft 13 April 2005”, http://www.wsmo.org/TR/d2/v1.2/D2v1-2_20050414.pdf. Accessed 01 Aug 2015
124. Ross TJ, “Fuzzy Logic with Engineering Applications”, 3rd Edition, *John Wiley & Sons, Ltd*, Chester, UK, pp 48-73, 2010
125. Ryoo J, Rizvi S, Aiken W and Kissell J, “Cloud Security Auditing: Challenges and Emerging Approaches”, *IEEE Security & Privacy*, vol. 12, no. 6, pp. 68-74, 2014
126. Sabou M and Fernandez M, “Ontology (Network) Evaluation. Ontology Engineering in a Networked World”, *Springer*, pp. 193-212, 2012
127. Salesfroce, CRM Software & Online, http://www.salesforce.com/uk. Accessed 02 Aug 2015
128. Sandikkaya MT and Harmanci AE, “Security Problems of Platform-as-a-Service (PaaS) Clouds and Practical Solutions to the Problems”, IEEE 31st Symposium on Reliable Distributed Systems (SRDS), pp. 463-468, 2012
129. Sellami M, Tata S, Maamar Z and Defude B, “A Recommender System for Web Services Discovery in a Distributed Registry Environment”, *Proceedings of Fourth IEEE International Conference on Internet and Web Applications and Services*, pp. 418-423, 2009
130. Serrano M, Shi L, Foghlú MÓ and Donnelly W, “Cloud Services Composition Support by Using Semantic Annotation and Linked Data*”, Knowledge Engineering and Knowledge Management, Communications in Computer and Information Science*, vol. 348, pp 278-293, 2013

131. Shearer R, Motik B and Horrocks I, “HermiT: A Highly Efficient OWL Reasoner”, *5th OWL Experienced and Directions Workshop*, pp. 26-27, 2008
132. Shen J, Beydoun G, Low G and Wang L, “Aligning Ontology-based Development with Service Oriented Systems”, *Future Generation Computer Systems*, vol. 32, pp. 263-273, 2014
133. Sheth A and Ranabahu A, “Semantic Modelling for Cloud Computing, Part 1”, *IEEE Internet Computing*, vol. 14, no. 3, pp.81-83, 2010
134. Sheth A and Ranabahu A, “Semantic Modelling for Cloud Computing, Part 2”, *IEEE Internet Computing*, vol. 14, no. 4, pp.81-84, 2010
135. Slack N, Chambers S, Johnston R and Betts A, “Operation and Process Management”, 2nd Edition, *Pearson Education Limited*, Essex, pp. 127-134, 2009
136. Smith MK, Welth C and McGuinness DL, “OWL Web Ontology Language Guide, W3C Recommendation”, http://www.w3.org/TR/owl-guide/. Accessed 10 Aug 2015
137. Sotomayor B, Montero Ruben S, Llorente IM and Foster I, “Virtual Infrastructure Management in Private and Hybrid Clouds”, *IEEE Internet Computing*, vol. 13, no. 5, pp. 14-22, 2009
138. Sousa L, Leite J and Loques O, “Green data centers: Using Hierarchies for Scalable Energy Efficiency in Large Web Clusters”, *Information Processing Letters*, vol. 113, no 14-16, pp 507-515, 2013
139. Stanoevska-Slabeva K and Wozniak T, “Cloud Basics – An Introduction to Cloud Computing, Grid and Cloud Computing”, *Springer Berlin Heidelberg*, pp. 47-61, 2010
140. Stoilos G, Stamou G, Pan JZ, Tzouvaras V and Horrocks I, “Reasoning with Very Expressive Fuzzy Description Logics”, *Journal of Artificial Intelligence Research*, vol. 30, pp. 273-320, 2007
141. Stoilos G, Stamou G and Pan JZ, “Fuzzy Extensions of OWL: Logical Properties and Reduction to Fuzzy Description Logics”, *International Journal of Approximate Reasoning*, vol. 51, no. 6, pp. 656-679, 2010
142. Suchithra R, Selvarani R and Nagamalai D, “Elements of Cloud Computing: A Perspective on Service Oriented Enterprises (SOEs)”, *Advances in Digital Image Processing and Information Technology, Communications in Computer and Information Science, Springer Berlin Heidelberg*, pp. 366-377, 2011

143. Sycara K and Paolucci M, “Dynamic Discovery and Coordination of Agent-based Semantic Web Services”, *IEEE Internet Computing*, vol. 8, no. 3, pp. 66-73, 2004
144. Tahamtan A, Beheshti SA, Anjomshoaa A and Tjoa AM, “A Cloud Repository and Discovery Framework Based on a Unified Business and Cloud Service Ontology”, *IEEE World Congress on Services (SERVICES)*, pp. 203-210, 2012
145. Tahir A, Tosi D and Morasca S, “A Systematic Review on the Functional Testing of Semantic Web Services”, *Journal of Systems and Software*, vol. 86, no. 11, pp. 2877-2889, 2013
146. Tari Z, “Security and Privacy in Cloud Computing”, *IEEE Cloud Computing*, vol. 1, no. 1, pp. 54-57, 2014
147. The Dasein Cloud API. http://dasein-cloud.sourceforge.net/. Accessed 26 Jun 2015
148. Topology and Orchestration Specification for Cloud Applications Version 1.0. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html. Accessed 22 Jul 2015
149. Toosi AN, Calheiros RN, Buyya R, “Interconnected Cloud Computing Environments: Challenges, Taxonomy, and Survey”. *ACM Computing Surveys*, vol. 47, no. 1.7, 2014
150. TOSCA Overview. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca#overview. Accessed 27 Jul 2015
151. Tserpes K, Aisopos F, Kyriazis D and Varvarigou T, “A Recommender Mechanism for Service Selection in Service-oriented Environment”, *Future Generation Computer Systems*, vol. 28, no. 8, pp. 1285-1294, 2012
152. Tsarkov D and Horrocks I, “FaCT++ Description Logic Reasoner: System Description”, *3rd International joint conference on Automatic Reasoning*, pp. 292-297, 2006
153. UDDI, http://uddi.xml.org/. Accessed 27 May 2014
154. Uddin M and Rahman AA, “Energy Efficiency and Low Carbon Enabler Green IT Framework for Data Centers Considering Green Metrics, Renewable and Sustainable Energy Reviews”, vol. 16, no. 6, pp. 4078-4094, 2012
155. Vhahn V, Santos L, Scoboria K, Scoboria E and Yeoh J, “Secaas Implementation Guidance: Web Security”, *Cloud Security Alliance*,

https://downloads.cloudsecurityalliance.org/initiatives/secaas/SecaaS_Cat_3_Web_Security_Implementation_Guidance.pdf. Accessed 19 Aug 2015.

156. Vhahn V, Santos L, Scoboria K, Scoboria E and Yeoh J, “Secaas Implementation Guidance: Security Assessments”, *Cloud Security Alliance*, https://downloads.cloudsecurity alliance.org/initiatives/secaas/SecaaS_Cat_5_Security_Assessments_Implementation_Guidance.pdf. Accessed 19 Aug 2015.
157. Vhahn V, Santos L, Scoboria K, Scoboria E and Yeoh J, “Secaas Implementation Guidance: Encryption”, *Cloud Security Alliance*, https://downloads.cloudsecurityalliance.org/initiatives/secaas/SecaaS_Cat_8_Encryption_Implementation_Guidance.pdf. Accessed 19 May 2015.
158. Vhahn V, Santos L, Scoboria K, Scoboria E and Yeoh J, “Secaas Implementation Guidance: Network Security Implementation Guidance”, *Cloud Security Alliance*, https://downloads.cloudsecurityalliance.org/initiatives/secaas/SecaaS_Cat_10_Network_Security_Implementation_Guidance.pdf. Accessed 19 Aug 2015.
159. VMunteanu V, Mindruta C and Fortis T, “Service Brokering in Cloud Governance”, *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC),* pp.497-504, 2012
160. Vaquero LM, Rodero-Merino L, Caceres J and Lindner M, “A Break in the Clouds: towards a Cloud Definition”, *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50-55, 2009
161. Wang X., Hauswirth M, Vitvar T and Zaremba M, “Semantic Web Services Selection Improved by Application Ontology with Multiple Concept Relations”, *Proceedings of the 2008 ACM symposium on Applied computing*, pp. 2237-2242, 2008
162. Wlodarczyk TW, Rong C, O'connor M and Musen M, “SWRL-F: a Fuzzy Logic Extension of the Semantic Web Rule Language”, *International Conference on Web Intelligence, Mining and Semantics*, no. 39, 2011
163. WSDL, http://www.w3.org/TR/wsdl. Accessed 19 Jul 2015
164. Xero, Features, http://www.xero.com/accounting-software. Accessed 15 Aug 2015
165. Xiao Z, Song W and Chen Q, “Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1107-1117, 2013

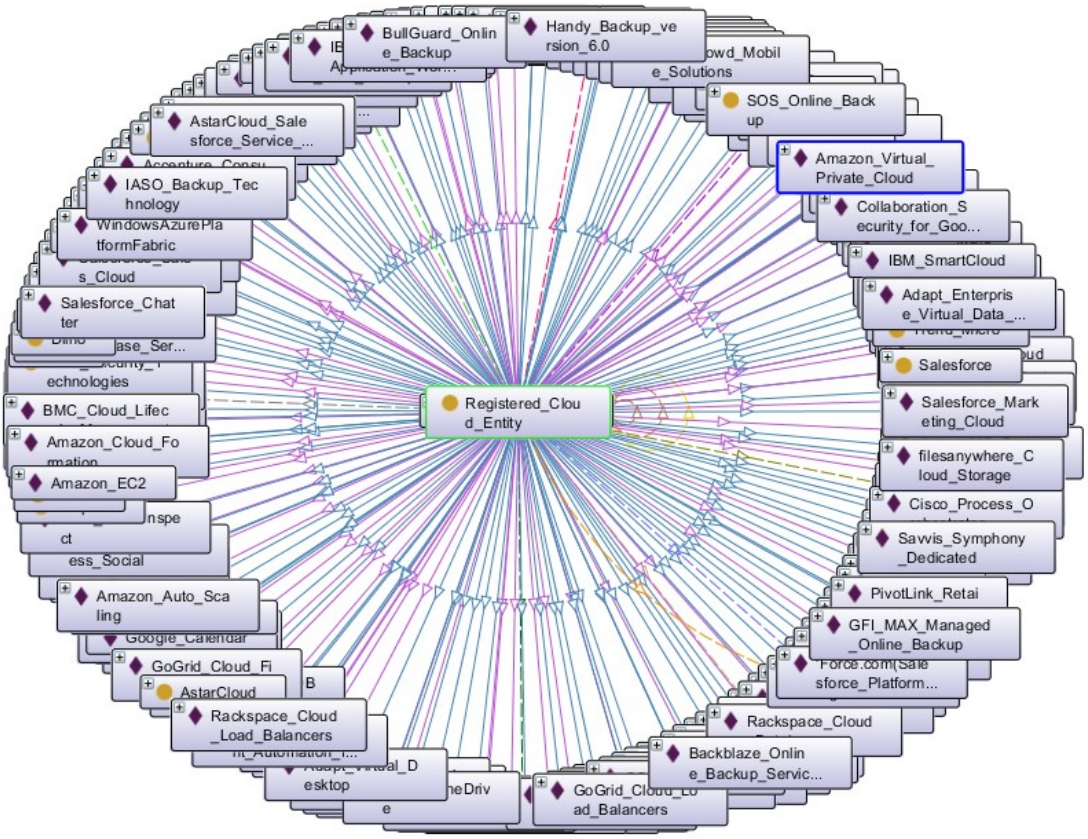
166. Yeh T and Lee H, “Enhancing Availability and Reliability of Cloud Data through Syncope*”, IEEE International Conference on Green Computing and Communications*, pp. 125-131, 2014
167. Youseff, L., Butrico, M. and Silva, D.D., “Toward a Unified Ontology of Cloud Computing”, Grid Computing Environments Workshop, *IEEE*, pp. 1-10, 2008.
168. Zedeh LA, “Fuzzy Set”, *Information and Control*, vol. 8, pp. 338-353, 1965
169. Zhang M, Ranjan R, Haller A, Georgakopoulos D, Menzel M and Nepal S, “An Ontology-based System for Cloud Infrastructure Services' discovery”, *8th International Conference on Collaborative Computing: Networking, Applications and Work sharing*, pp.524-530, 2012
170. Zhang Y, Li Y and Zheng W, “Automatic Software Deployment using User-level Virtualization for Cloud-computing”, *Future Generation Computer Systems*, vol. 29, no. 1, pp. 323-329, 2011
171. Zoho, Explore the Features of the Zoho Recruit, https://www.zoho.com/recruit/features.html. Accessed 10 Aug 2015

Appendix A Abbreviations and Acronyms

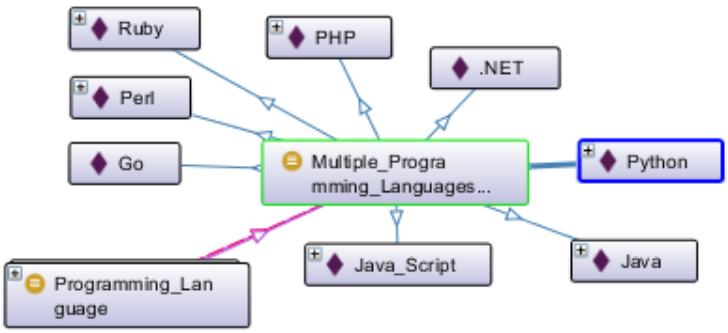
All the abbreviations and acronyms used in this thesis are defined below.

Abbreviation /Acronyms	Description
AoFeCSO	agility-oriented & fuzziness-embedded cloud service ontology
API	application programming interface
CC	cloud computing
CSAMO	cloud service access and manipulation ontology
CSC	cloud service consumer
CSI	cloud service instance
CSP	cloud service provider
CSRMP	cloud service recommendation and management platform
DL	description logic
DoS	denial of service
DP	datatype property
FL	fuzzy logic
IaaS	infrastructure-as-a-service
ICT	information communication technology
OS	operating system
OP	object property
OPM	operation process map
OWL	web ontology language
QoS	quality of service
PaaS	platform-as-a-service
PLN	probabilistic logic framework
PSSA	provider-specific service aspect
SaaS	software-as-a-service
SAMOS	service access and manipulation operation specification
SIR	service information request
SLA	service level agreement
SMR	service manipulation request
SOA	service oriented architecture
SOPMM	service operation process map modelling
UDDI	universal description, discovery and integration
WSDL	web service description language
WSMF	web service modelling framework

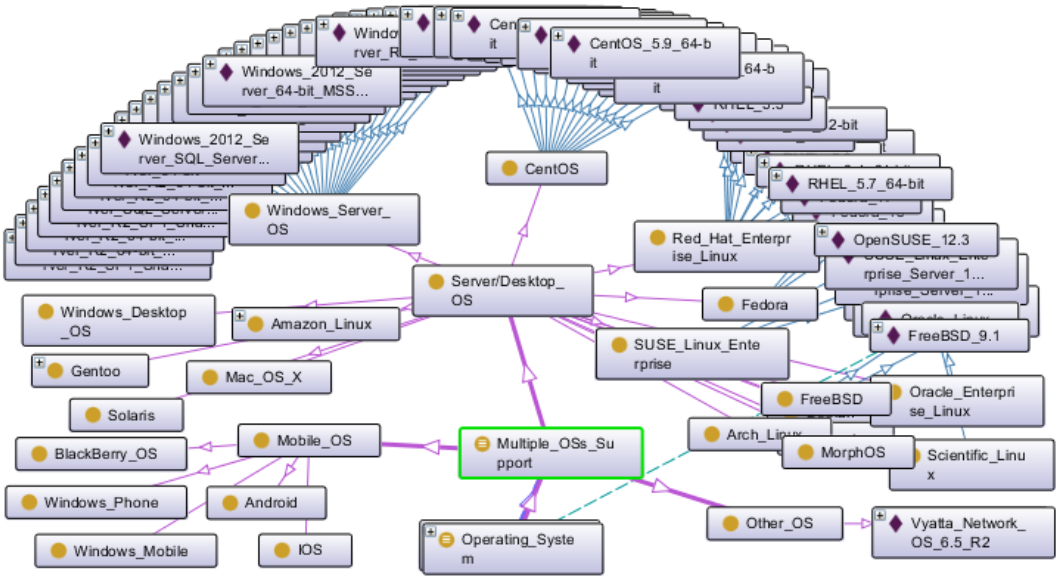
Appendix B AoFeCSO Entity Screenshots



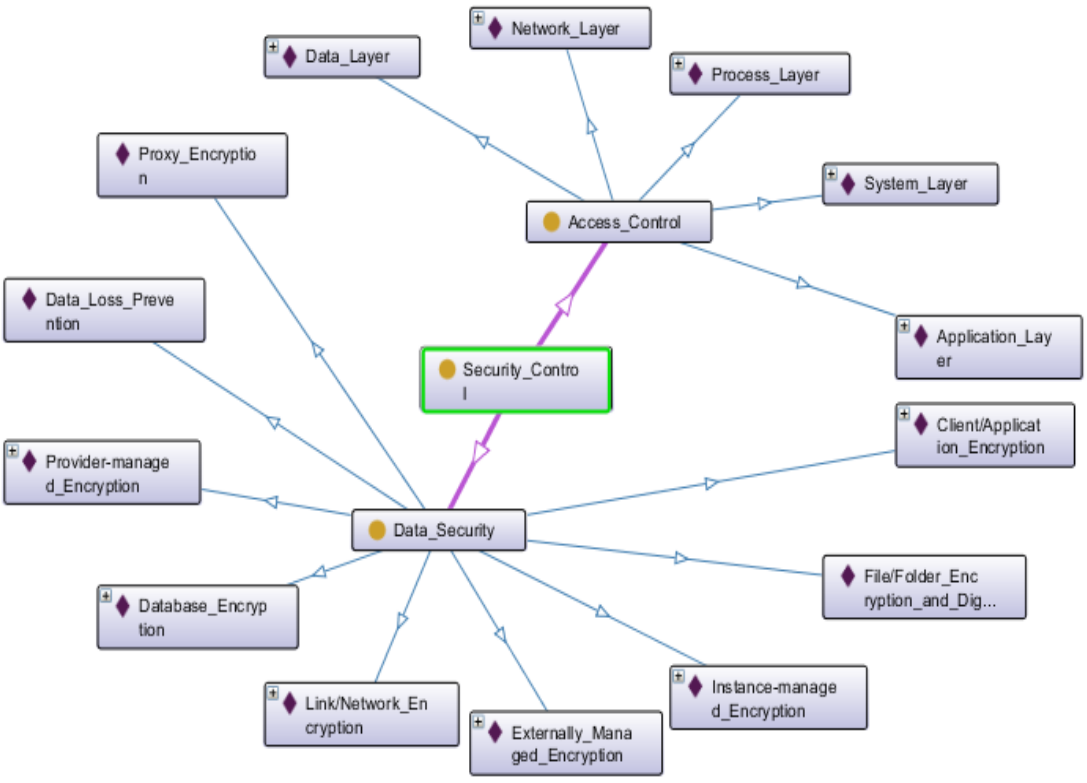
i. Cloud service entities



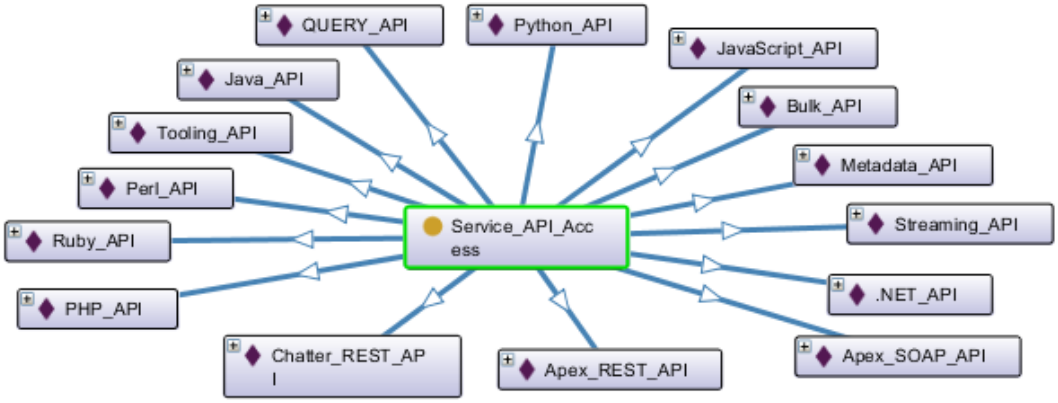
ii. Cloud service programming language support entities



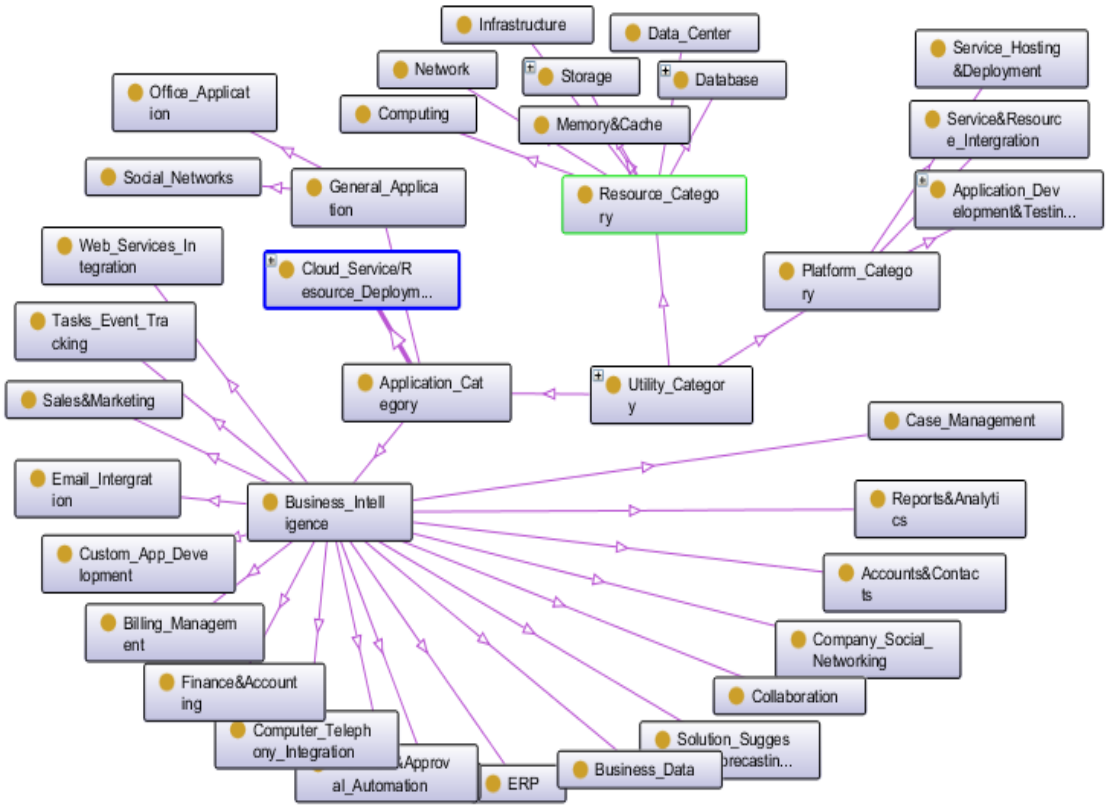
iii. Cloud service operating system support entities



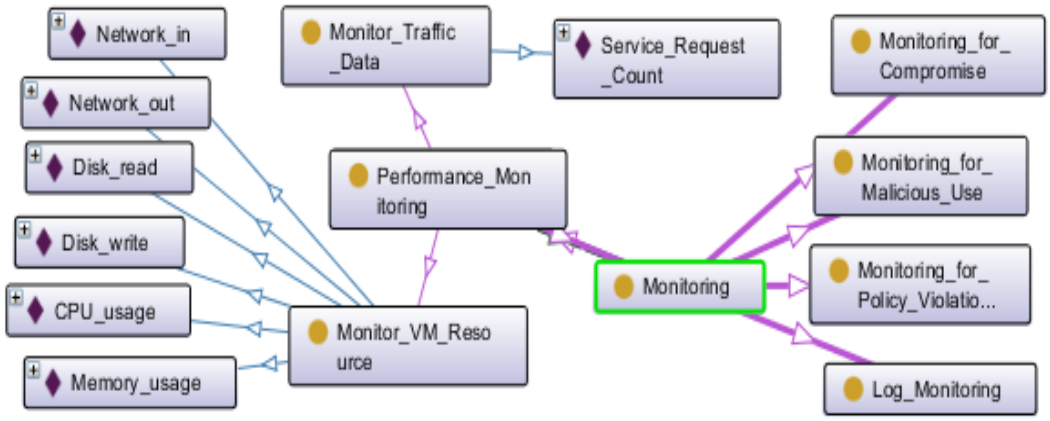
iv. Cloud service security entities



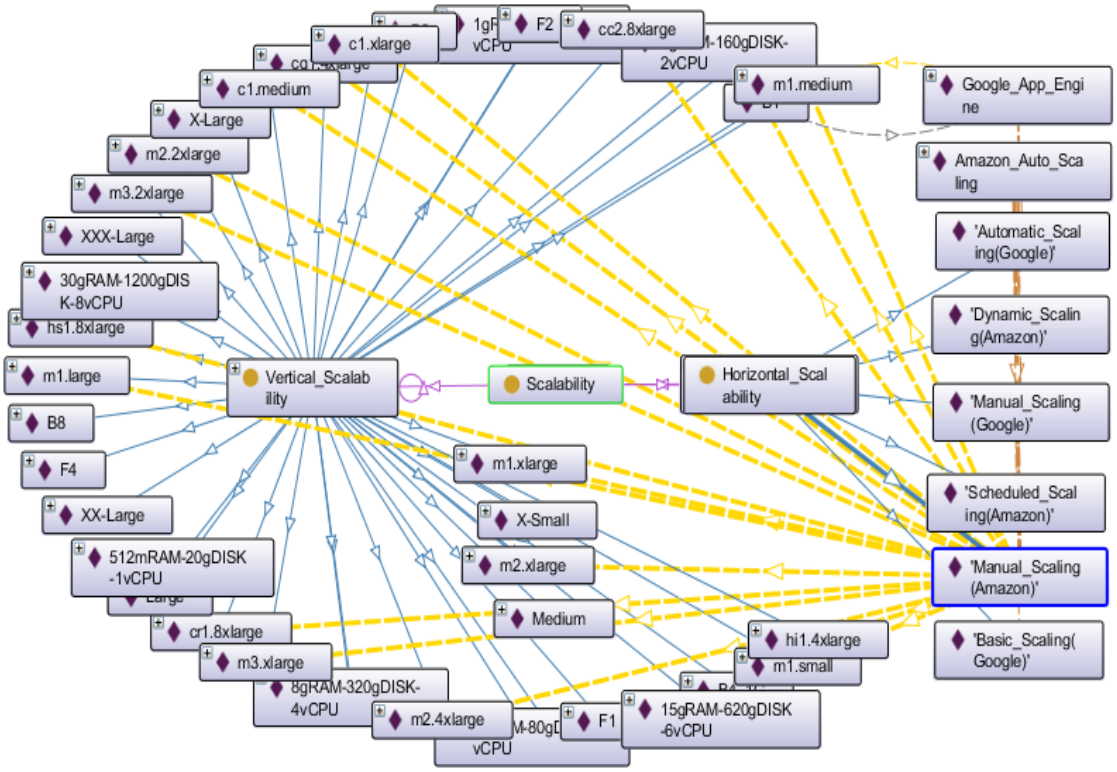
v. Cloud service API entities



vi. Cloud service function entities

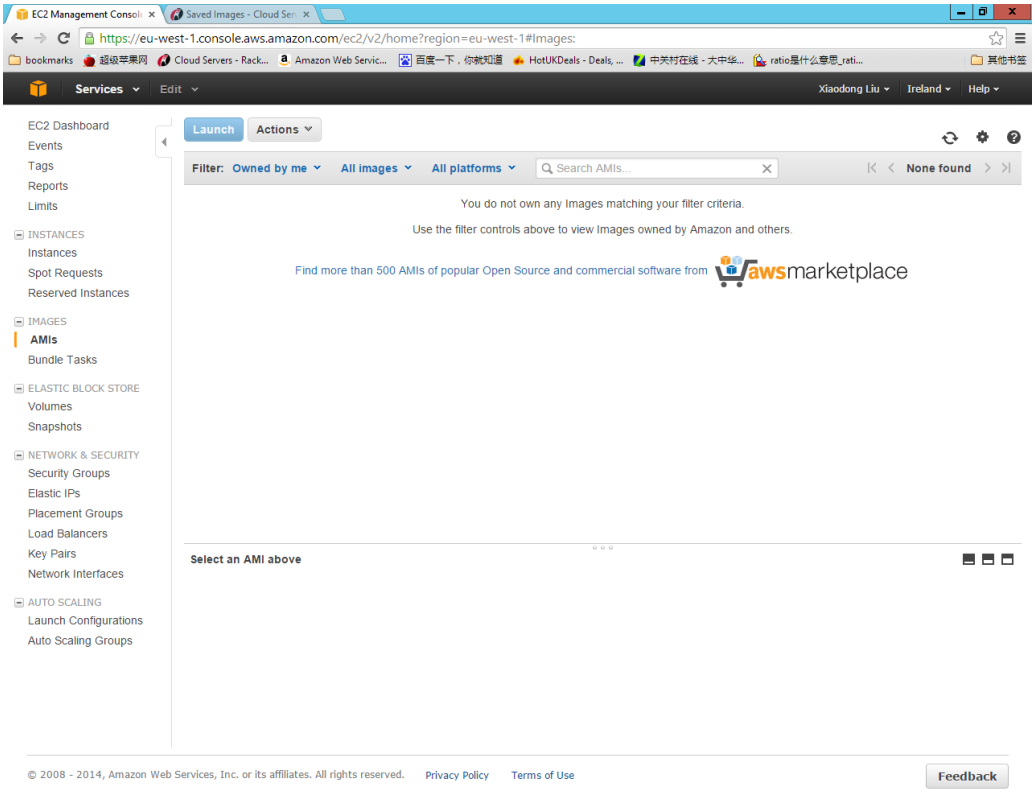


vii. Cloud service monitor entities

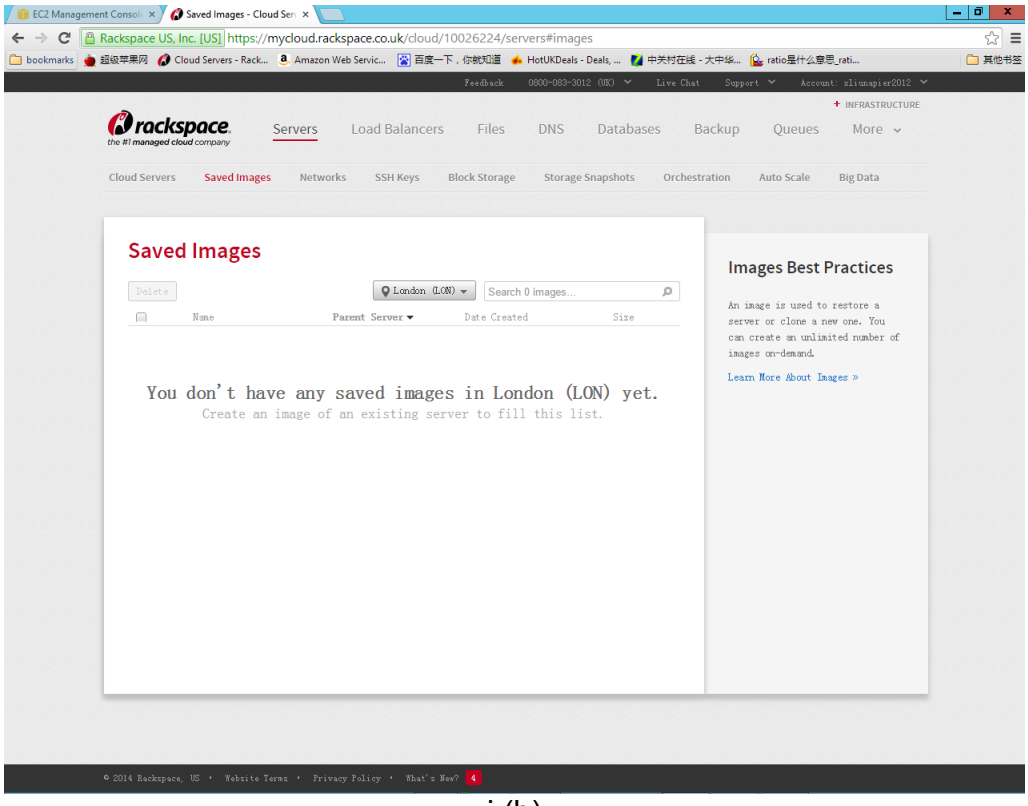


viii. Cloud service scalability entities

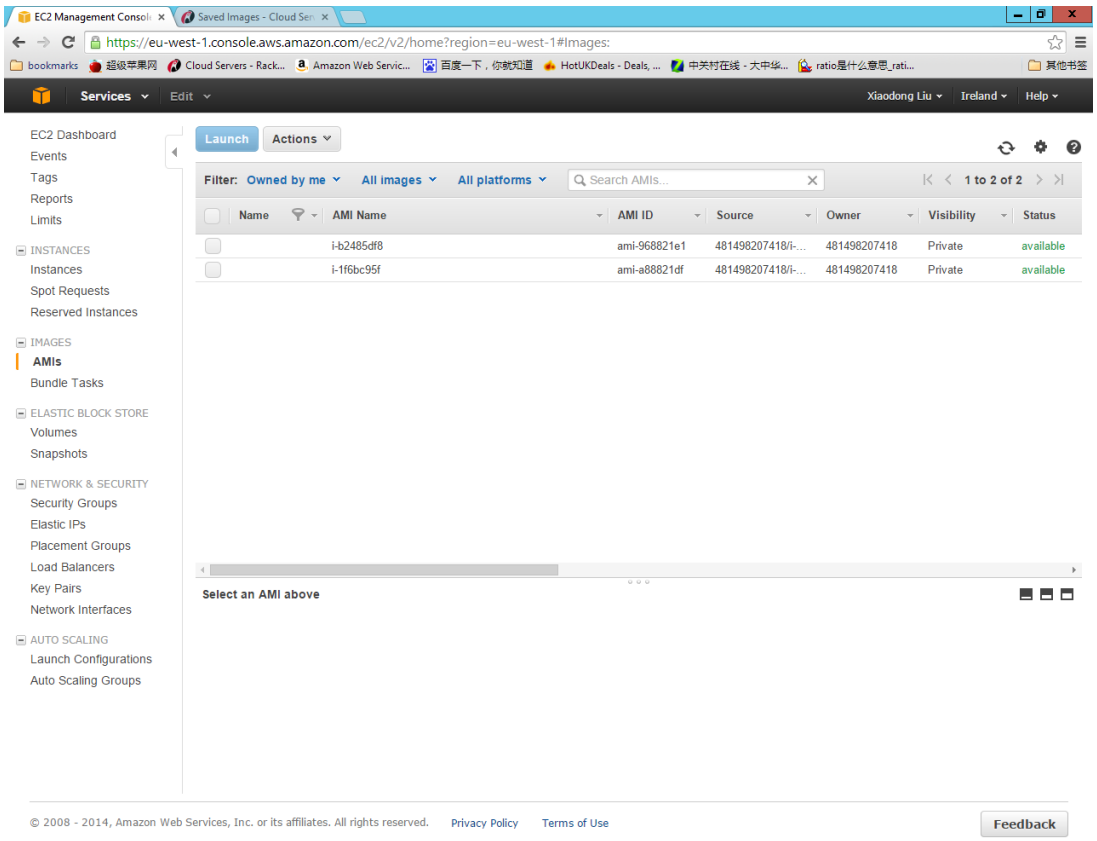
Appendix C Cloud Service Web Portal Screenshots



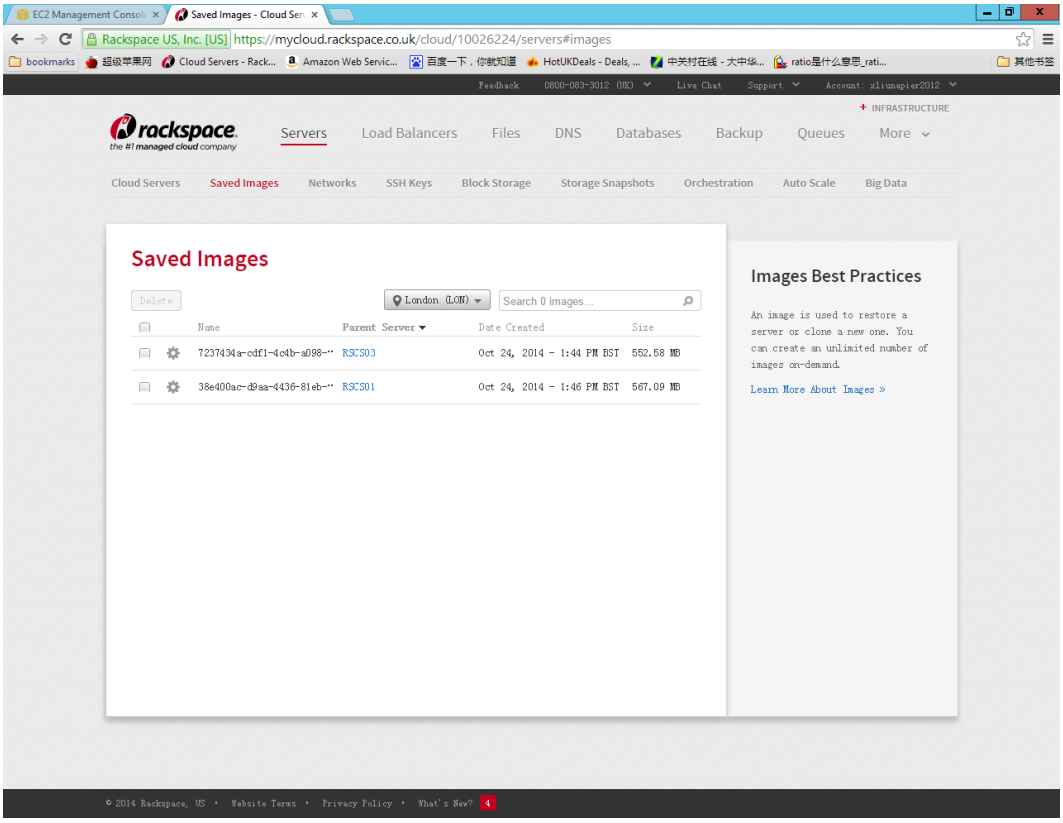
i.(a)



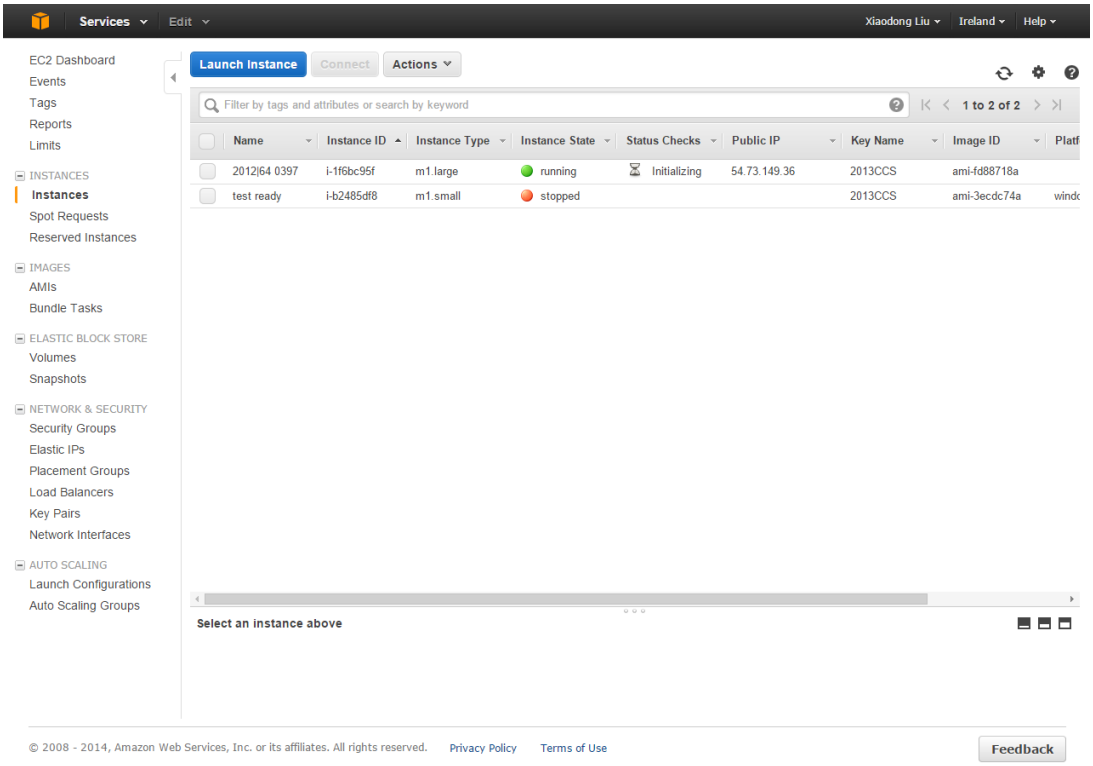
i.(b)


EC2 Management Consol
Saved Images - Cloud Ser
https://eu-west-1.console.aws.amazon.com/ec2/v2/home?region=eu-west-1#Images:
Services
Edit
Xiaodong Liu
Ireland
Help
EC2 Dashboard
Events
Tags
Reports
Limits
INSTANCES
Instances
Spot Requests
Reserved Instances
IMAGES
AMIs
Bundle Tasks
ELASTIC BLOCK STORE
Volumes
Snapshots
NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Load Balancers
Key Pairs
Network Interfaces
AUTO SCALING
Launch Configurations
Auto Scaling Groups
Launch
Actions
Filter: Owned by me
All images
All platforms
Search AMIs...
1 to 2 of 2
Name
AMI Name
AMI ID
Source
Owner
Visibility
Status
i-b2485df8
ami-968821e1
481498207418/i-...
481498207418
Private
available
i-1f6bc95f
ami-a88821df
481498207418/i-...
481498207418
Private
available
Select an AMI above
© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Feedback

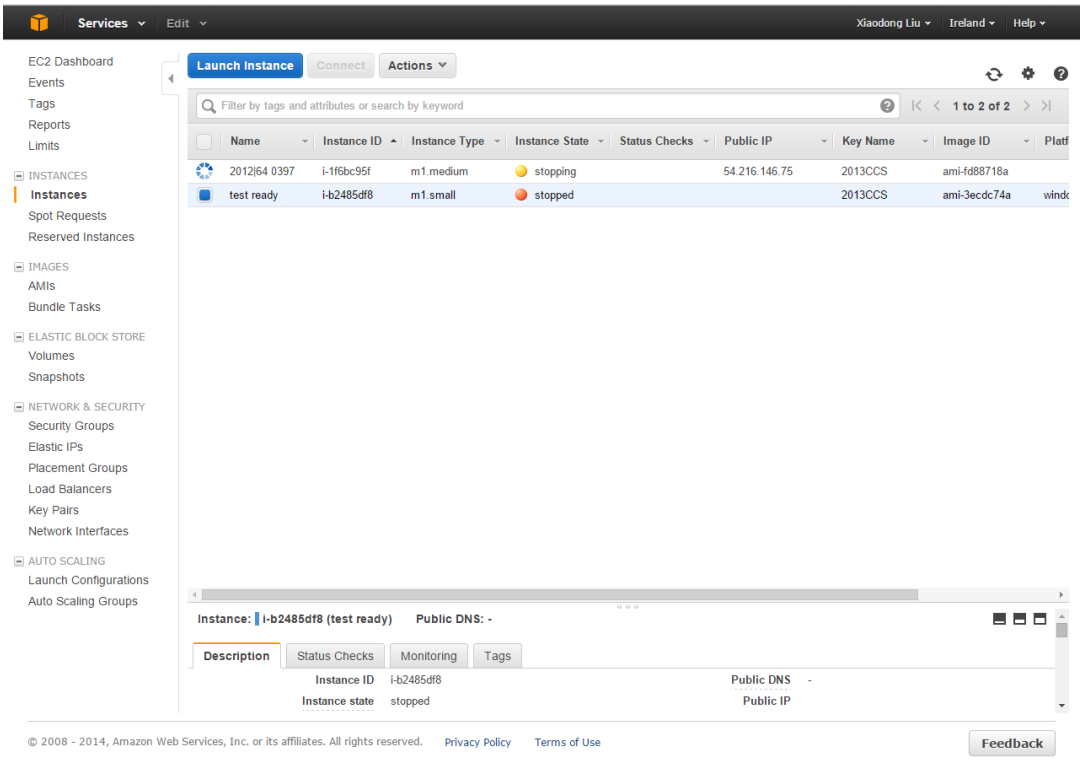
i.(c)


EC2 Management Consol
Saved Images - Cloud Ser
Rackspace US, Inc. [US]
https://mycloud.rackspace.co.uk/cloud/10026224/servers#images
Feedback
0800-083-3012 (UK)
Live Chat
Support
Account: xliunapier2012
INFRASTRUCTURE
rackspace
the #1 managed cloud company
Servers
Load Balancers
Files
DNS
Databases
Backup
Queues
More
Cloud Servers
Saved Images
Networks
SSH Keys
Block Storage
Storage Snapshots
Orchestration
Auto Scale
Big Data
Saved Images
Delete
London (LON)
Search 0 images...
Name
Parent Server
Date Created
Size
7237434a-cdf1-4c4b-a098-...
RSCS03
Oct 24, 2014 - 1:44 PM BST
552.58 MB
38e400ac-d9aa-4436-81eb-...
RSCS01
Oct 24, 2014 - 1:46 PM BST
567.09 MB
Images Best Practices
An image is used to restore a server or clone a new one. You can create an unlimited number of images on-demand.
Learn More About Images »
© 2014 Rackspace, US
Website Terms
Privacy Policy
What's New?
4

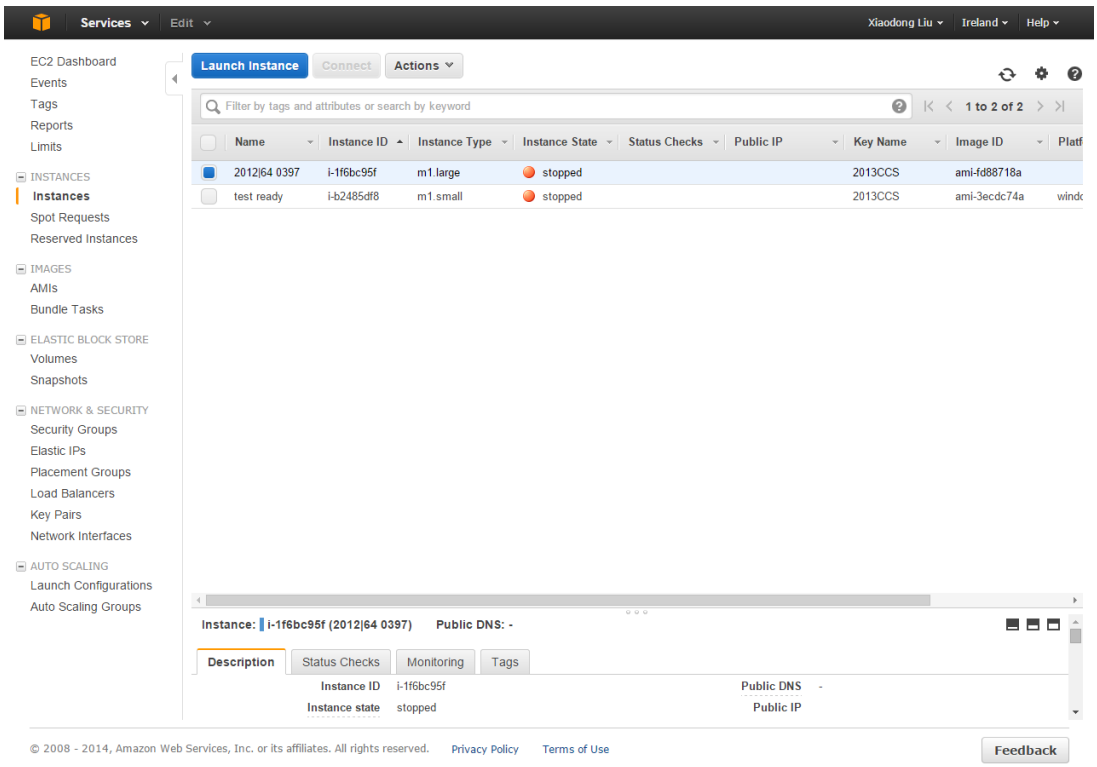
i.(d)


Services
Edit
Xiaodong Liu
Ireland
Help
EC2 Dashboard
Events
Tags
Reports
Limits
INSTANCES
Instances
Spot Requests
Reserved Instances
IMAGES
AMIs
Bundle Tasks
ELASTIC BLOCK STORE
Volumes
Snapshots
NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Load Balancers
Key Pairs
Network Interfaces
AUTO SCALING
Launch Configurations
Auto Scaling Groups
Launch Instance
Connect
Actions
Filter by tags and attributes or search by keyword
1 to 2 of 2
Name
Instance ID
Instance Type
Instance State
Status Checks
Public IP
Key Name
Image ID
2012|64 0397
i-1f6bc95f
m1.large
running
Initializing
54.73.149.36
2013CCS
ami-fd88718a
test ready
i-b2485df8
m1.small
stopped
2013CCS
ami-3ecdc74a
Select an instance above
© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Feedback

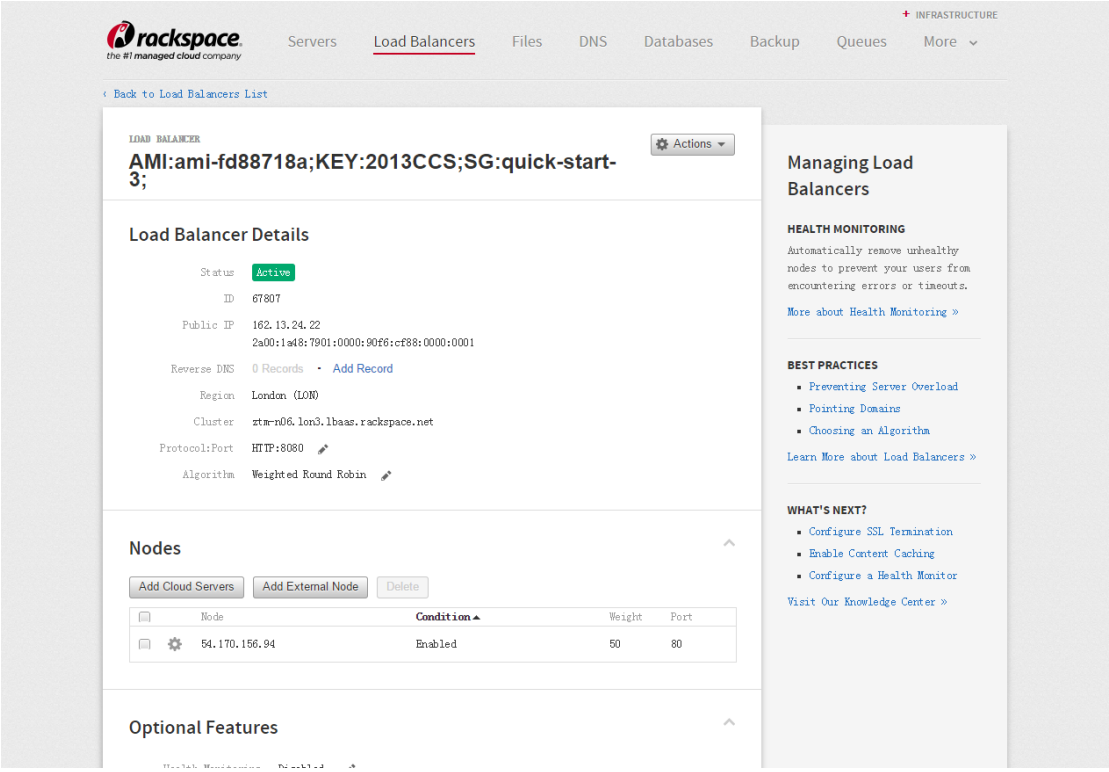
ii.(a)


Services
Edit
Xiaodong Liu
Ireland
Help
EC2 Dashboard
Events
Tags
Reports
Limits
INSTANCES
Instances
Spot Requests
Reserved Instances
IMAGES
AMIs
Bundle Tasks
ELASTIC BLOCK STORE
Volumes
Snapshots
NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Load Balancers
Key Pairs
Network Interfaces
AUTO SCALING
Launch Configurations
Auto Scaling Groups
Launch Instance
Connect
Actions
Filter by tags and attributes or search by keyword
1 to 2 of 2
Name
Instance ID
Instance Type
Instance State
Status Checks
Public IP
Key Name
Image ID
2012|64 0397
i-1f6bc95f
m1.medium
stopping
54.216.146.75
2013CCS
ami-fd88718a
test ready
i-b2485df8
m1.small
stopped
2013CCS
ami-3ecdc74a
Instance: i-b2485df8 (test ready)
Public DNS: -
Description
Status Checks
Monitoring
Tags
Instance ID
i-b2485df8
Instance state
stopped
Public DNS
Public IP
© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Feedback

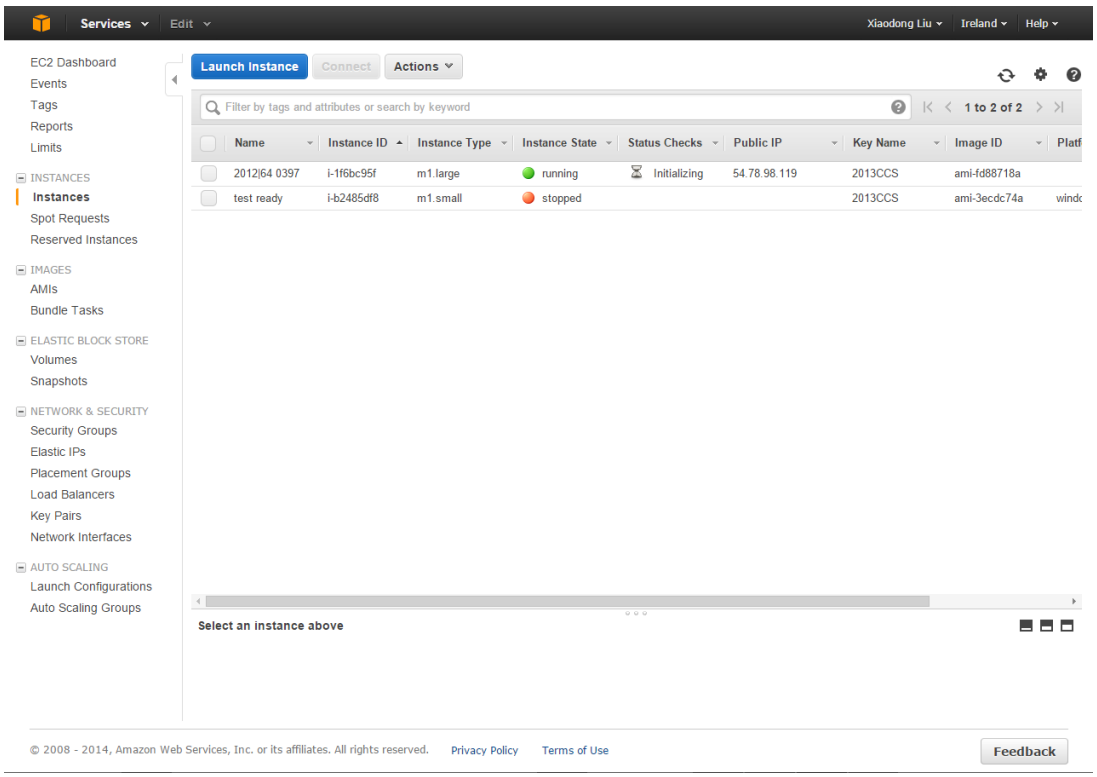
ii.(b)


Services
Edit
Xiaodong Liu
Ireland
Help
EC2 Dashboard
Events
Tags
Reports
Limits
INSTANCES
Instances
Spot Requests
Reserved Instances
IMAGES
AMIs
Bundle Tasks
ELASTIC BLOCK STORE
Volumes
Snapshots
NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Load Balancers
Key Pairs
Network Interfaces
AUTO SCALING
Launch Configurations
Auto Scaling Groups
Launch Instance
Connect
Actions
Filter by tags and attributes or search by keyword
1 to 2 of 2
Name
Instance ID
Instance Type
Instance State
Status Checks
Public IP
Key Name
Image ID
2012|64 0397
i-1f6bc95f
m1.large
stopped
2013CCS
ami-fd88718a
test ready
i-b2485df8
m1.small
stopped
2013CCS
ami-3ecdc74a
Instance: i-1f6bc95f (2012|64 0397)
Public DNS: -
Description
Status Checks
Monitoring
Tags
Instance ID
i-1f6bc95f
Public DNS
Instance state
stopped
Public IP
© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Feedback

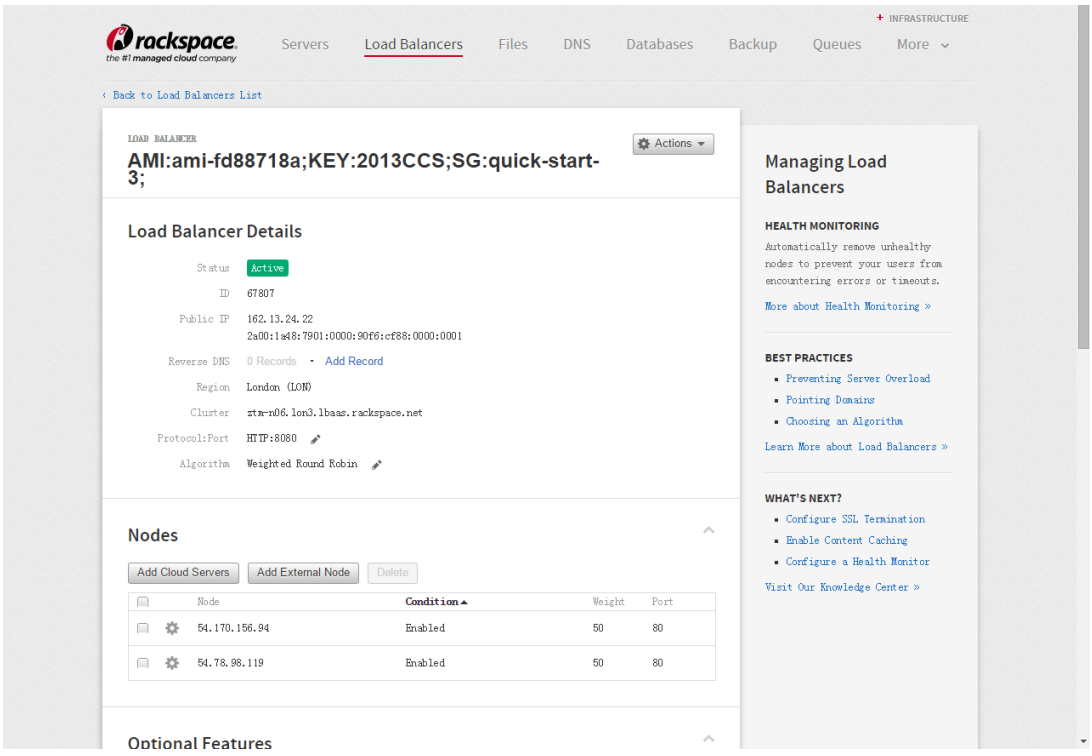
iii.(a)


rackspace
the #1 managed cloud company
INFRASTRUCTURE
Servers
Load Balancers
Files
DNS
Databases
Backup
Queues
More
Back to Load Balancers List
LOAD BALANCER
AMI:ami-fd88718a;KEY:2013CCS;SG:quick-start-3;
Actions
Load Balancer Details
Status
Active
ID
67807
Public IP
162.13.24.22
2a00:1a48:7901:0000:90f6:cf88:0000:0001
Reverse DNS
0 Records
Add Record
Region
London (LON)
Cluster
ztm-n06.lon3.lbaas.rackspace.net
Protocol:Port
HTTP:8080
Algorithm
Weighted Round Robin
Nodes
Add Cloud Servers
Add External Node
Delete
Node
Condition
Weight
Port
54.170.156.94
Enabled
50
80
Optional Features
Managing Load Balancers
HEALTH MONITORING
Automatically remove unhealthy nodes to prevent your users from encountering errors or timeouts.
More about Health Monitoring »
BEST PRACTICES
Preventing Server Overload
Pointing Domains
Choosing an Algorithm
Learn More about Load Balancers »
WHAT'S NEXT?
Configure SSL Termination
Enable Content Caching
Configure a Health Monitor
Visit Our Knowledge Center »

iii.(b)


Services
Edit
Xiaodong Liu
Ireland
Help
EC2 Dashboard
Events
Tags
Reports
Limits
INSTANCES
Instances
Spot Requests
Reserved Instances
IMAGES
AMIs
Bundle Tasks
ELASTIC BLOCK STORE
Volumes
Snapshots
NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Load Balancers
Key Pairs
Network Interfaces
AUTO SCALING
Launch Configurations
Auto Scaling Groups
Launch Instance
Connect
Actions
Filter by tags and attributes or search by keyword
1 to 2 of 2
Name
Instance ID
Instance Type
Instance State
Status Checks
Public IP
Key Name
Image ID
2012|64 0397
i-1f6bc95f
m1.large
running
Initializing
54.78.98.119
2013CCS
ami-fd88718a
test ready
i-b2485df8
m1.small
stopped
2013CCS
ami-3ecdc74a
Select an instance above
© 2008 - 2014, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Feedback

iii.(c)


rackspace
the #1 managed cloud company
INFRASTRUCTURE
Servers
Load Balancers
Files
DNS
Databases
Backup
Queues
More
Back to Load Balancers List
LOAD BALANCER
AMI:ami-fd88718a;KEY:2013CCS;SG:quick-start-3;
Actions
Load Balancer Details
Status
Active
ID
67807
Public IP
162.13.24.22
2a00:1a48:7901:0000:90f6:cf88:0000:0001
Reverse DNS
0 Records
Add Record
Region
London (LON)
Cluster
ztm-n06.lon3.lbaas.rackspace.net
Protocol:Port
HTTP:8080
Algorithm
Weighted Round Robin
Nodes
Add Cloud Servers
Add External Node
Delete
Node
Condition
Weight
Port
54.170.156.94
Enabled
50
80
54.78.98.119
Enabled
50
80
Optional Features
Managing Load Balancers
HEALTH MONITORING
Automatically remove unhealthy nodes to prevent your users from encountering errors or timeouts.
More about Health Monitoring »
BEST PRACTICES
Preventing Server Overload
Pointing Domains
Choosing an Algorithm
Learn More about Load Balancers »
WHAT'S NEXT?
Configure SSL Termination
Enable Content Caching
Configure a Health Monitor
Visit Our Knowledge Center »

iii.(d)

Appendix D PaaS Service/CSI/PSSA operation specifications for AWS Elastic Beanstalk

Cloud Service Level Operations	***Elastic Beanstalk***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SR PostCondition**
List Applications	SIR	Unconditional	ElasticBeanstalk Region(M)	ElasticBeanstalk ApplicationName (s)	Unconditional
List Application Environment	SIR	Unconditional	ElasticBeanstalk Region(M)	ElasticBeanstalk EnvironmentID(s)	Unconditional
Delete Application	SMR	Unconditional	ElasticBeanstalk ApplicationName(M)	Operation Succeeded	Unconditional
Delete Application Environment	SMR	Unconditional	ElasticBeanstalk EnvironmentID(M)	Operation Succeeded	Unconditional

CSI Level Operations	***Elastic Beanstalk Application Instance***				
	Type	**SRPreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SR PostCondition**
Get Application Environment	SIR	Unconditional	ElasticBeanstalk ApplicationName(M)	ElasticBeanstalk EnvironmentID	Unconditional
Get Application Versions	SIR	Unconditional	ElasticBeanstalk ApplicationName(M)	ElasticBeanstalk ApplicationVersionDescrptions	Unconditional
Create Application	SMR	Unconditional	Elastic Beanstalk ApplicationName(M), Elastic Beanstalk ApplicationDescription(O)	ElasticBeanstalk ApplicationName	Elastic Beanstalk EnvironmentStatus is in "Ready" state
Update Application	SMR	Elastic Beanstalk EnvironmentStatus is in "Ready" state	Elastic Beanstalk ApplicationName(M), Elastic Beanstalk ApplicationDescription(O)	ElasticBeanstalk ApplicationName	Elastic Beanstalk EnvironmentStatus is in "Ready" state

PSSA Level Operations	***Elastic Beanstalk Application Environment***				
	Type	**SR PreCondition**	**SRParameter/SRSubject**	**SROutcome**	**SR PostCondition**
Get Application Environment VMs	SIR	Unconditional	ElasticBeanstalk EnvironmentID (M)	EC2 InstanceIDs	Unconditional
Get Application Environment LoadBalancers	SIR	Unconditional	ElasticBeanstalk EnvironmentID (M)	Elastic LoadBalancer ID	Unconditional
Create Application Environment	SMR	Unconditional	ElasticBeanstalk ApplicationName(M), ElasticBeanstalk EnvironmentDescription(O), ElasticBeanstalk EnvironmentName(M), Elastic Beanstalk ConfigurationOptionSettings<…>(O), etc.	ElasticBeanstalk EnvironmentID	Unconditional
Update Environment Configuration	SMR	Elastic Beanstalk EnvironmentStatus is in "Ready" state	Elastic Beanstalk ConfigurationOptionSettings<…>(M)	ElasticBeanstalk EnvironmentID	Elastic Beanstalk EnvironmentStatus is in "Ready" state