A Semantic-Oriented Description Framework and Broker Architecture for Publication and Discovery in Cloud Based Conferencing

Jerry George

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science : “Software Engineering” at

Concordia University

Montréal, Québec, Canada

July 2013

ABSTRACT

A Semantic-Oriented Description Framework and Broker Architecture for Publication and Discovery in Cloud-Based Conferencing

Jerry George

Cloud computing is an emerging paradigm for provisioning network, storage, and computing resources on demand using a pay-per-use model. Conferencing is the conversational exchange of media between several parties. Cloud-based conferencing services can provide benefits such as easy introduction of different types of conferences, resource usage efficiency and scalability.

A business model has been recently proposed in a position paper for cloud-based conferencing with the following roles: conference substrate provider, conference infrastructure provider, conference platform provider, conference service provider, and broker. Conference substrates are generally atomic and served as elementary building blocks (e.g. signaling, mixing) of conferencing applications. They can be virtualized and shared for resource efficiency purposes. Multiple conferencing substrates can be combined to build a conferencing service (e.g. a dial-out audio signaling conference service composed from dial-out signaling and audio mixer substrates).

The focus of this thesis is to design a semantic-oriented description framework for conferencing substrates and an architecture for their publication and discovery. The description framework is made up of a description language and a cloud-based conference ontology. The conference ontology is modeled on the basis of the interacting roles in the proposed cloud-based conferencing business model. The overall publication and discovery architecture for cloud-based conference substrates is made up of three brokers and the related publication and discovery interfaces. The publication and discovery interfaces are modelled using REpresentation State Transfer (REST) interfaces. A prototype is built to demonstrate the feasibility of this architecture. The effectiveness of the architecture is also proved using the performance measurements.

ACKNOWLEDGEMENT

First, I offer my deepest gratitude and appreciation to my supervisor Dr. Roch Glitho for all his support, assistance and guidance throughout the course of my research. This work would not have been possible without his immense patience, continuous guidance, and countless drafts reviewed. It was a pleasure working and learning from him throughout the period of research at Concordia University.

I also offer my sincere gratitude to Dr. Fatna Belqasmi and Dr. Nadjia Kara for all her ideas, continuous feedback and their effort and time put in. Their countless patience and enthusiasm helped me a lot along the way for the successful completion of my research work.

I'm grateful to Dr. Goswami and Dr. Tsantalis for serving as members of my thesis committee and for their valuable comments and ideas. I am also grateful to Dr. G. Butler for chairing the thesis review.

I would like to take the opportunity to thank my colleague Flora Taheri for all her ideas, help and countless drafts reviewed of the thesis work. I would also like to thank my colleagues and friends at the Telecommunication Services Engineering Research Lab for their advices, ideas, and helps.

I would also like specially thank to Dr. Roch Glitho and Concordia University for their financial support during the period of my research assistantship. Last but not least I would like to thank and praise my dearest family, especially my wife and my son, for their invaluable patience and confidence in me and supporting me during the completion of my degree.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

3G	Third Generation
3GPP	Third Generation Partnership Project
AHP	Analytical Hierarchical Process
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BGP	Basic Graph Pattern
BOSH	Bidirectional-streams Over Synchronous HTTP
CIP	Conference Infrastructure Provider
CLI	Command Line Interface
CSMIC	Cloud Service Measurement Index Consortium
CSP	Conference Substrate Provider
CSV	Comma Separated Values
DAML-S	DARPA Agent Markup Language for Services
DARPA	Defense Advanced Research Projects Agency
DC	Dublin Core
DL	Description Logics
ETSI	European Telecommunications Standards Institute
FOAF	Friend of a Friend
GUI	Graphical User Interface
GUID	Globally Unique Identifier
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol

IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
InfGraph	Inference Graph
IOPE	Input-Output-Preconditions-Effects
JAX-RS	Java APIs for RESTful Web Services
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation - Linked Data
kbps	Kilo Bits Per Second
LAMP	Linux, Apache Web Server, MySQL database and PHP
Linked USDL	Linked Unified Service Description Language
MCDM	Multi-Criteria Decision-Making
MEP	Message Exchange Pattern
MSM	Minimal Service Model
N3	Notation 3
NIST	National Institute of Standards and Technology
OntoQL	Ontology Query Language
OWL	Web Ontology Language
OWL-S	Web Ontology Language for Services
PaaS	Platform as a Service
PSL	Process Specification Language
QoS	Quality Of Service
RAM	Random Access Memory

RAM	Random Access Memory
RDBMS	Relational Database Management System
RDF	Resource Description Framework
RDFa	Resource Description Framework in Attributes
RDFS	Resource Description Framework Schema
REST	Representational State Transfer
RFC	Request for Comments
RSS	Really Simple Syndication
RTCP	Real Time Control Protocol
SaaS	Software as a Service
SA-REST	Semantic Annotations for REST
SA-WSDL	Semantically Annotated Web Service Description Language
SD	Substrate Description
SESA	Semantically Enabled Service Oriented Architectures
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPARQL	SPARQL RDF Query Language
STI	Semantic Technology Institute
SWS	Semantic Web Services
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
USDL	Unified Service Description Language

W3C	World Wide Web Consortium
WADL	Web Application Description Language
WSDL	Web Service Description Language
WSM	Weighted Sum Model
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
WSMX	Web Service Modeling eXecution environment
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

Chapter 1

1 Introduction

This chapter provides an introduction to the key research areas and an overview of the related concepts. It discusses the motivation, the problem statement, and the salient contributions of this thesis. Finally, it gives an outline of how this thesis document is organized.

1.1 Definitions

1.1.1 Conferencing

Conferencing is the real-time multi-party exchange of media (voice, video, and text). Multi-party conferencing is ubiquitous nowadays and enables real-time collaboration between conference participants. Conferencing presents itself as a significant component of several applications such as large scale enterprise applications, gaming, social networking applications, etc.

1.1.2 Cloud Computing

According to National Institute of Standards and Technology (NIST) [1] cloud computing is an emerging and transformational paradigm for provisioning of network, storage, and computing on demand as a commodity using a pay-per-use model. Cloud computing has several inherent benefits such as scalability, efficiency of resource utilization, reliability, easier management, and a coherent and flexible pricing model. According to one of the popular and widely-accepted definitions of cloud computing [1], it encompasses three key facets: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). IaaS provides the infrastructure for computation, storage and networking using virtualized hardware resources. PaaS provides the software environments to design, develop, test, deploy and maintain applications. SaaS provides software applications and composite services to end-users or other applications.

1.1.3 Cloud Conferencing Business Model

Due to the benefits of cloud computing, there is a growing trend towards the migration of different types of applications to the cloud computing landscape. However, to the best of our knowledge, there are currently no full-fledged environments that allow the development, deployment and management of cloud-based conferencing applications [2]. In this vein, a business model was proposed for cloud-based conferencing [3]. The following are the roles in the proposed business model: connectivity provider, broker, conferencing substrate provider, conferencing infrastructure provider, conferencing platform provider, and conferencing service provider. Virtualized conference substrates (e.g. dial-out signaling, audio mixing) are sharable fine-grained building blocks of conferencing provided by the conference substrate provider. These virtualized conference substrates can be composed to create full-fledged conferencing applications (e.g. dial-out audio conference)

1.2 Motivations and Problem Statement

Cloud-based conferencing services present a promising use-case for conferencing applications with significant benefits such as easy introduction of different types of conferences, as well as benefits inherited from cloud computing such as resource efficiency, and scalability.

For the realization of cloud-based conferencing, it is critical to identify a mechanism for describing virtualized conference substrates and design an architecture for efficient publication and discovery of the conference substrates in a cloud setting. In this architecture, a key role is played by the conference substrate providers. Conference substrate providers publish the re-usable and sharable virtualized conference substrates to a broker. The conference infrastructure provider discovers these conference substrates from the broker and may re-publish them as they are, or the conference infrastructure provider may alternatively choose to combine them with other substrates and then publish the resulting substrates. The conference service provider uses the conference platform provider to discover the conference substrates provided by the conference infrastructure provider and assemble these conference substrates to build different types of conference applications on the fly and publish them to a broker. Later, the conference end-users discover these conferencing applications created by the conference service provider. This architecture permits the interaction between all the roles in the proposed cloud-based conferencing business model for the purpose of publication and discovery of conference substrates.

1.3 Thesis Contributions

The thesis contributions are as follows:

- Requirements for a semantic-oriented description framework and a broker architecture that enables the interactions between the roles in the proposed cloud conferencing business model.
- Analysis of the state of the art with an evaluation summary based on our requirements.

- An overall architecture for publication and discovery of cloud-based conferencing substrates based on the business model.
- Implementation architecture, a proof of concept prototype, and performance evaluation.

1.4 Thesis Organization

The remaining sections are divided into six chapters as follows:

Chapter 2 presents the background concepts and definitions related to the research domain in more detail. Concepts such as conferencing, cloud computing, semantic web, and brokers are explained.

Chapter 3 presents the requirements and evaluation of the state of the art related to semantic description framework and broker architecture for publication and discovery in cloud-based conferencing.

Chapter 4 presents the proposed architecture for publication and discovery in cloud-based conferencing. It describes the proposed semantic-oriented description framework for conference substrates. It also describes architecture for the broker and explains its components in detail.

Chapter 5 describes the implementation architecture and technologies used for the proof-of-concept prototype. Also, it provides information regarding the developed benchmarking tool for measurements. At the end of this chapter, the observed performance measurements are presented.

Chapter 6 concludes the thesis by giving a summary of the overall contributions and future research directions.

Chapter 2

2 Background

This chapter describes background concepts related to the research domain. The following concepts are explained: cloud computing architecture, conferencing, semantic web, and broker.

2.1 Cloud Computing Architecture

In this section we start by giving a definition of cloud computing and its key benefits. After that, we explain the key facets of cloud computing.

2.1.1 Definition and Key Benefits of Cloud Computing

According to the NIST, cloud computing is an emerging and transformational paradigm for provisioning of network, storage, and computing on demand as a commodity (using a pay-per-use model) [1]. Cloud computing focuses on shifting the allocation of resources (e.g. infrastructure, platform and software) to a transparent network to reduce the cost associated with purchasing dedicated hardware and software solutions [4].

Some of the key benefits [5], [6] of cloud computing are as follows:

- Scalability
- Resource efficiency
- Reliability
- Resource pooling

- Easier management of resources
- Coherent and flexible price model (on-demand or pay-as-you-go)

2.1.2 Key Facets of Cloud Computing

Cloud Computing consists of the following key facets [4]:

- **Infrastructure as a Service (IaaS)** providing an infrastructure to compute, store, and network using virtualized hardware abstractions.
- **Platform as a Service (PaaS)** providing software environments to design, build, test, deploy, host, and maintain applications and services.
- **Software as a Service (SaaS)** providing full-fledged applications and services to end-users and other applications using APIs.

Figure 1 illustrates the three key facets of cloud computing architecture with some examples. In this sub-section we will explain each of these three key facets of cloud computing in detail.

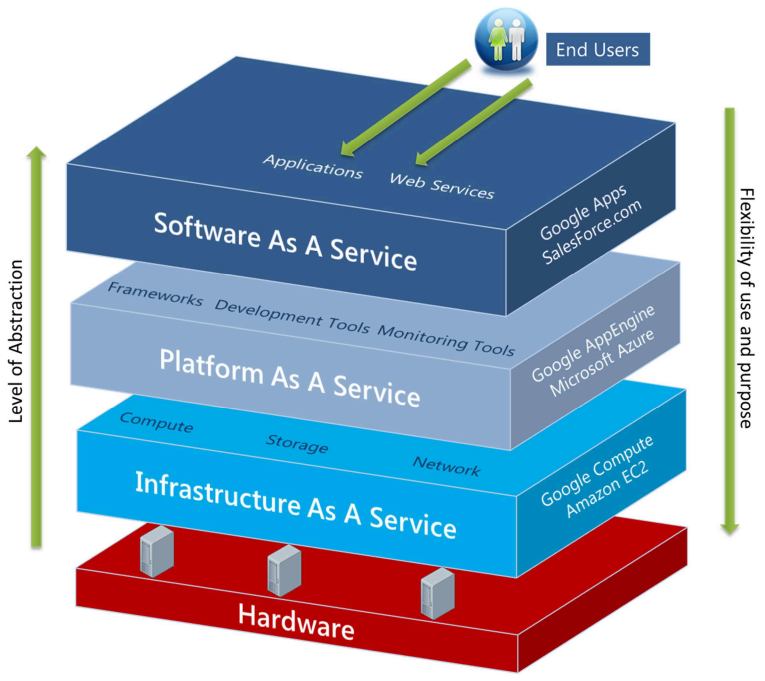


Figure 1: Cloud Computing Architecture [6]

2.1.2.1 Infrastructure as a Service (IaaS)

IaaS includes a dynamic pool of virtualized computing, storage, and network resources. Virtualization technology enables the co-existence of multiple heterogeneous resources including network, storage, and computing. Hence, it provides us benefits such as greater cost efficiency through less hardware requirements. A software tool called Hypervisor or Virtual Machine Manager is used to make best use of the hardware resources by maintaining and monitoring the virtualized resources. In the traditional scenario, resources would have been managed by hardware engineers and system administrators involving a huge investment from enterprises. Examples of infrastructure providers include Amazon, Elastic Cloud 2 (EC2), Google Compute Engine, and Rackspace Cloud.

2.1.2.2 Platform as a Service (PaaS)

PaaS includes the platform for service providers for designing, implementing, testing, hosting, deploying, and maintaining the applications and services. It includes components such as Application Virtual Machines or runtime environments such as Java Virtual Machine (JVM), Linux, Apache Web Server, MySQL database and PHP (LAMP) to enable the development and deployment of applications. Examples of PaaS providers include Google App Engine, Engine Yard, Couchbase, and Windows Azure App Fabric.

2.1.2.3 Software as a Service (SaaS)

SaaS includes applications provided by service providers to end-users directly or to other third-party applications via APIs. Service providers charge end-users using pay-per-use models. One of the key advantages from the perspective of end-users is the absence of capital expenditure for resources (hardware, software licensing). Examples of SaaS providers include Tropo, Salesforce.com, DocuSign, and oDesk.

2.2 Conferencing

In this section, we give a brief introduction to conferencing, key technical components of a conference, and different types of conferences. Later, we discuss the Cloud-based conferencing business model [3].

2.2.1 A Brief Introduction to Conferencing

Conferencing is the real-time multi-party exchange of media (voice, video, and text) which enables real-time collaboration with varying degrees of interactivity among the conference participants. Conferencing is ubiquitous nowadays and presents itself as a significant component of several applications such as large-scale enterprise applications, gaming applications and social networking applications. Conference services are resource-intensive and require efficient real-time processing capabilities, as they could involve several hundred users around the globe with different types of devices.

Conferencing has been extensively studied by standard bodies such as Internet Engineering Task Force (IETF), Third Generation Partnership Project (3GPP), and European Telecommunications Standards Institute (ETSI). The IETF XCON working group has published a framework for conferencing [7] , a floor control protocol [8], and has developed an information data model for conferencing [9]. 3GPP defines a specification for IP Multimedia Subsystem (IMS), which provides multimedia services to end-users in a 3G Network. In part, this specification provides an architecture [10] and APIs [11] for multimedia conferencing.

2.2.2 Key Technical Components of a Conference

A typical conference consists of the following key technical components (depicted in Figure 2) [3][7],

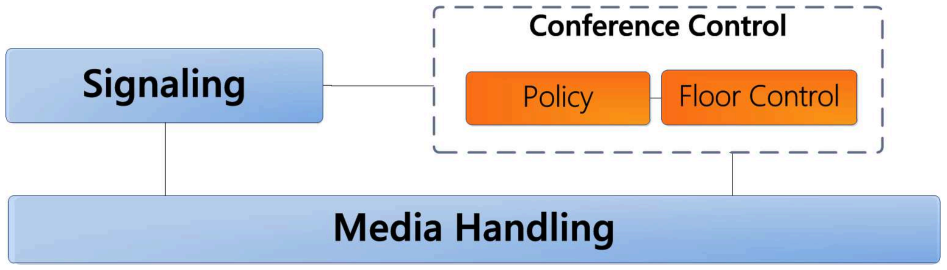


Figure 2 : Conference Architecture

- **Signaling** –The signaling component handles operations such as session set-up, capability negotiation, and session tear down. Signaling protocols such as Session Initiation Protocol (SIP) and Jabber are usually used.
- **Media Handling** – The media handling component manages media aspects such as transmission, mixing and transcoding. The mixer is an entity combining multiple input audio/video streams into a single output stream. The mixer generates multiple output streams for each participant to ensure that participants receive only the streams from the other participants present in a conference, and not the stream from themselves. Transcoding is the process of encoding and decoding between different media formats. Depending on the transcoding capability of the media handling component, the conference application may or may not support a specific device type.
- **Conference Control –** Conference control provides advanced functionalities such as floor control and policy control [3]. Conference control is an optional component in the conference architecture.
    - **Policy Control –** The policy control component handles conference policy aspects such as conference and participant management, admission control, etc.

- **Floor Control –** The floor control component allows joint or exclusive access to specific resources provisioned as part of a conference.

For any type of conference application, signaling and media handling form the most critical parts. For instance, a simple dial-out audio conference consists of dial-out signaling and audio mixer media handling components.

2.2.3 Different Types of Conferences

RFC 4353 standard specification "A Framework for Conferencing with the Session Initiation Protocol" [12] defines three major types of conferencing. They are defined as follows:

- **Tightly-coupled Conference** – Tightly coupled conference is a conference in which there is an entity called focus that hosts a conference and maintains the signaling relationships with all the participants. The focus plays the key role of the centralized manager of the conference, and is addressed by a conference URI.
- **Fully-distributed Conference** - Fully distributed conference is a conference where each participant maintains a signaling relationship with all of the other participants. Similar to the loosely coupled conference, distributed conference also does not have a centralized manager; management is completely distributed amongst its participants.
- **Loosely-coupled Conference –** Loosely-coupled Conference is a conference without a signaling relationship between every participant in conference. The participants learn about each other through multicast protocols using the Real Time Control Protocol (RTCP).

2.2.4 Cloud-based Conferencing Business Model

A cloud-based conferencing business model [3] was recently proposed with a vision for provisioning (planning, implementing, deploying, executing, managing, and monitoring) complex conference applications in the cloud setting. This model allows easy introduction of different types of conferences and flexibility of provider selection (preventing vendor lock-ins) in addition to taking full advantage of benefits of cloud computing. Figure 3 depicts the business model in detail with the key actors.

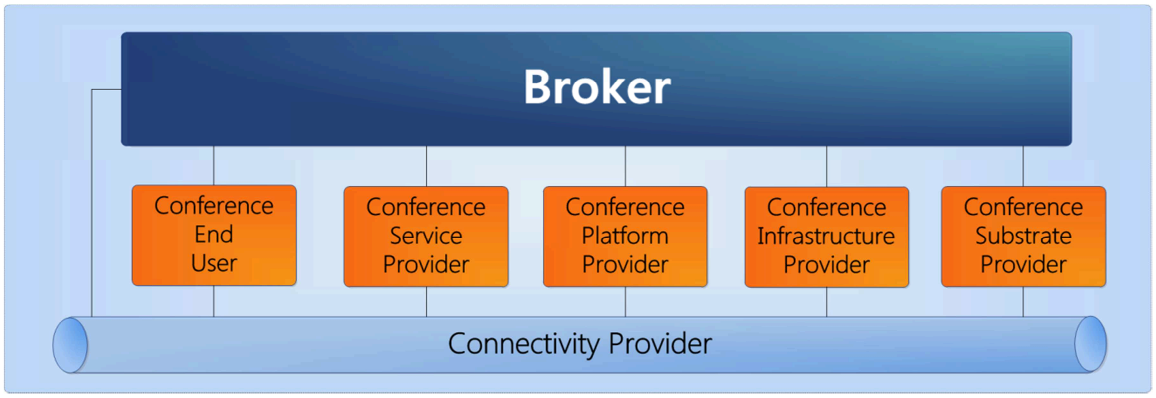

Broker
Conference End User
Conference Service Provider
Conference Platform Provider
Conference Infrastructure Provider
Conference Substrate Provider
Connectivity Provider

Figure 3: Cloud Conference Business Model

The key actors in the proposed business model are as follows:

- **Connectivity Provider –** The connectivity provider acts as communication channel for interactions between the different providers and requesters in the model.
- **Broker –** The broker provides a publication and discovery intermediary in different levels for substrates and conference applications. It also provides algorithms for the selection of the most appropriate substrate or conference application.

- **Conference Substrate Provider** – The substrate provider provides virtualized fine-grained conference substrates which constitute the main building blocks of conference applications.
- **Conference Infrastructure Provider** – The infrastructure provider discovers the substrates provided by the substrate provider via the broker. Alternatively, it may also combine its own substrates and publish them to the broker for discovery by the conference platform provider tools.
- **Conference Platform Provider** – The platform provider provides a set of tools for creation, composition, and execution of conference services by providing software frameworks and service logic execution environments for conference applications. It discovers the substrates provided by the infrastructure provider via the broker to be used in conference applications.
- **Conference Service Provider –** The service provider utilizes the exposed platform tools provided by the platform provider in order to design and build conference applications. Once the conference applications are created, the application is published to the broker.
- **Conference End User –** The conference end user discovers the conference applications of interest from the broker and subscribes to them using pay-per-use model.

2.3 Semantic Web

According to the W3C , the Semantic Web provides a common framework that contains a set of technologies and specifications that allow data to be shared and reused across applications, enterprises, and community boundaries [13]. Semantic Web with its constituent technologies and specifications can be classified into four layers. In this section we give a brief introduction to

Semantic Web. Next, we introduce the Semantic Web Layers and the key technologies and specifications (at each layer) that are pertinent to our research.

2.3.1 Brief Introduction to Semantic Web

Semantic Web is an extension of the World Wide Web. Tim Berners Lee coined the term Semantic Web in 2001 with two important goals: 1) To turn the Web into a collaborative medium to both share and aggregate data and services from heterogeneous sources [14] and 2) To make inter-operability of machines possible based on data and services. To realize these goals, we require a standard description infrastructure to semantically describe both data and services [15]. The data and services are often referred to as resources.

We use the concept of semantic annotations to provide additional descriptions to the data and services. The semantic annotations are backed by a formal description of the concepts of a particular domain. Such a structural representation of the concepts within a specific domain and the semantic relationship between the concepts is called an *ontology* [16]. Ontology languages such as OWL allow us model such concepts and semantic relationships.

2.3.2 Semantic Web Layers, Key Technologies and Specifications relevant to our research

The Semantic Web can be broadly classified into four layers as follows,

- Data and Metadata Layer
- Semantics Layer
- Enabling Technology Layer
- Environment Layer

Such a classification was initially proposed by K Breitman et al. [17]. There are several key technologies and specification that enable the realization of the goals of Semantic Web [17]. These technologies and specifications lay the foundation to the Semantic Web. The Figure 4 illustrates the Semantic Web Layers and the key constituent technologies and specifications that are relevant to our research. In the following sub-sections we briefly introduce these key technologies and specifications.

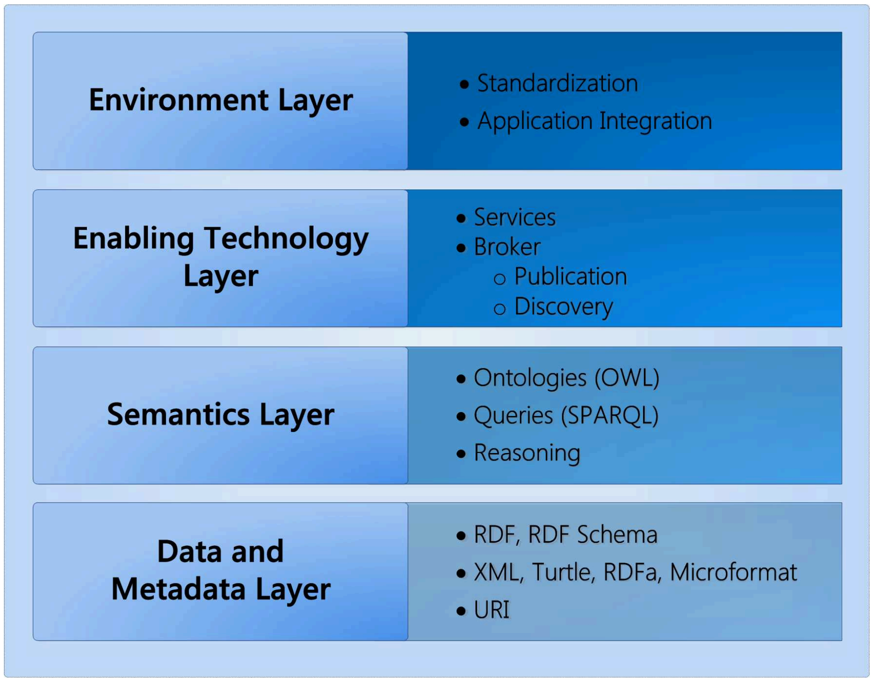


Figure 4: Semantic Web Layers[17]

2.3.2.1 Data and Metadata Layer

The data and metadata layer is the bottom-most layer providing the foundational elements to describe Web resources. The key elements of this layer are as follows:

- **Uniform Resource Identifier (URI)** is the de facto standard for uniquely identifying an abstract or physical resource. It is a string of characters conforming to US-ASCII (ASCII) encoding.
- **Resource Description Framework (RDF)** [18] is a standard description model for structuring information published on the Web. It is a fundamental building block of the Semantic Web. RDF allows us to define simple statements about the Web resources using triple (subject-predicate-object) as depicted below in Figure 5.

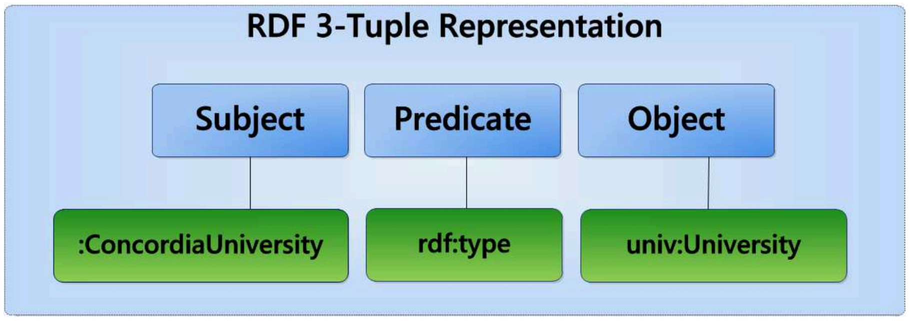


Figure 5: RDF Statement Structure

Assuming that we have university ontology with namespace prefix of "univ", we can define that Concordia University is a type of university. Each element of triple, except the object, always represents Web resources identified by a URI. An object can be either a Web resource or a literal value such as an integer or a string.

- **Extensible Markup Language (XML) and Turtle** are W3C-recommended serializations for describing structured data in RDF. Among these two, Turtle offers a human-readable format.

- **RDFa** is W3C recommendation for annotation of human-readable Web Pages for adding structured information. It allows HTML/XHTML documents to contain markups with mapping to specific RDF-based vocabulary by using standard attributes. Being XHTML-based, it allows both human (using browsers) and machine readability of structured data. Unlike most other mechanisms, it re-uses the statement structure created for RDF i.e. subject-predicate-object, and hence it provides a natural mapping between RDF and human-readable Web Pages.
- **Microformat** provides a similar mechanism as that of RDFa to annotate human-readable Web Pages. However, Microformat proposes a different set of attributes to annotate HTML/XHTML documents and is currently not accepted as a standard. There are also other syntaxes provided by W3C such as JSON-LD [19] which are used to represent structured data and provide translations to RDF.

- **RDF Scheme or RDF(S)** builds on top of RDF specifications and provides a set of classes and properties. RDFS provides a mechanism to relate properties to classes and define the domain and range by using properties. The domain of a property indicates for what type of classes the property is defined for. Whereas, the range of a property indicates the type of values (type of classes) that the property can assume.

2.3.2.2 Semantics Layer

The semantics layer is built on top of the data and metadata layer. It adds semantics to the resource representations using a set of specifications and technologies. Semantics layer enables machine interpretation, validation, reasoning, inference, and querying. Such specifications and technologies facilitate a machine to make autonomous decisions. The followings are some of the specifications and technologies at the semantics layer,

- **Web Ontology Language (OWL) -** Web Ontology Language (OWL) [20] is a logic-based ontology language proposed to help improve the interpretability of information by machines. OWL offers better and richer expressivity by providing additional vocabulary to the data and metadata layer specifications, namely RDF and RDF(S) [21]. The foundation for the logic in OWL comes from the Description Logics (DL) formalisms. DL is a set of formalisms for knowledge representation. DL provides knowledge in the form of three key characteristics: concepts (or classes), roles (or properties), and individuals (or instances). OWL provides formal semantics for describing ontology.

  Technically, ontology consists of a set of axioms and facts. Axioms allow us to create a formal definition for the classes and properties, whereas facts are instances based on the formal definitions. Classes are used define a collection of Web resources, whereas properties are used to provide relationships or define attributes on these Web resources.

  Illustrative Example of Axioms and Facts

  Figure 6 illustrates a set of class axioms such as `Student`, `Employee`, `Professor`, `Assistant`, `TeachingAssistant`, `ResearchAssistant`, and `Dissertation`. It also illustrates the property axioms such as `hasAdvisor`, and `writesDissertation`. `writesDissertation` has a domain of `ResearchAssistant` and a range of `Dissertation`. Based on these class axioms and property axioms, we define a set of facts or individuals on the right-hand side. Conceptually, the set of RDF triples forms a directed labeled graph as displayed in Figure 6. Hence, they represent structured data. Having such directed labeled graphs makes it more suitable for representation and interlinking of data and services [22].

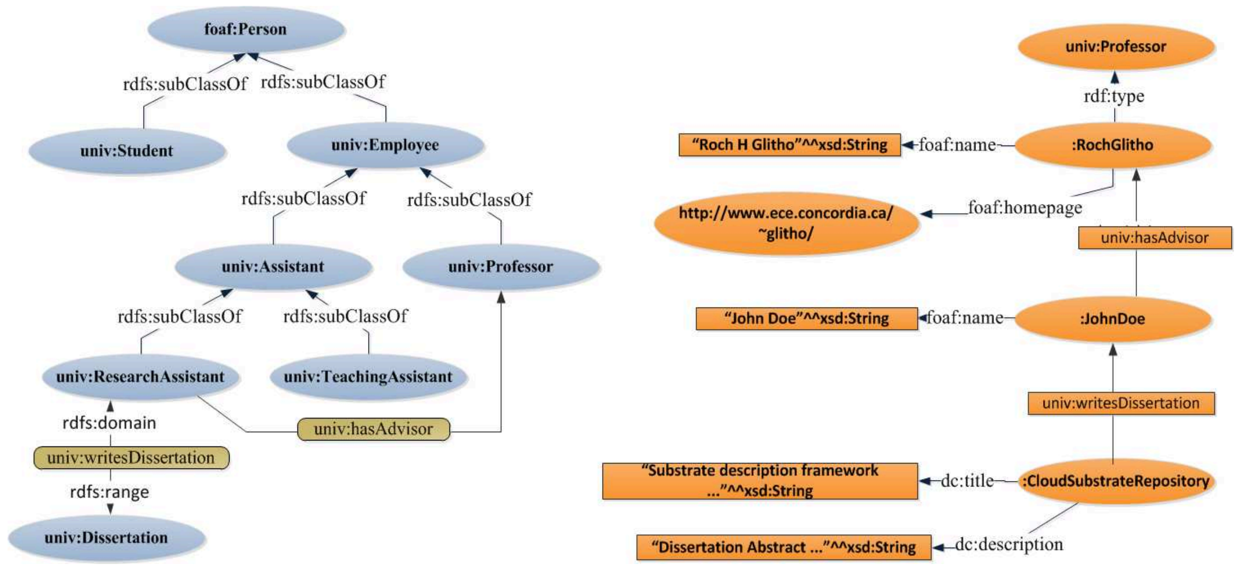


Figure 6: University Ontology – Axioms and Facts

Depending on the level of expressivity of OWL, three different sub languages of OWL v1.1 are defined: OWL Lite, OWL DL, and OWL Full. For a more detailed explanation of their comparison please refer to reference [20]. The flip-side of offering richer expressivity is the computational complexity required for reasoning and inference tasks.

- **Ontology-based Reasoning and Validation –** Reasoning and validation ensure that the facts or the instance of information are a logical consequence of the schema or ontology. The reasoning tools have an implicit sense of intelligence for generating new facts from existing facts. In Figure 6 the instance classes `:RochGlitho` and `:JohnDoe` can be inferred as employees through a transitive relationship even though they are not explicitly stated as part of the facts. Both of these instance classes belong to a subclass of `univ:Employee`. For performing such reasoning, there are several libraries and tools such as Jena, FaCT++, Pellet, HermiT, and RacerPro. A detailed comparison is available in reference [23].

- **SPARQL** [24] is a highly-expressive language for querying diverse RDF-based data sources. It is highly expressive and powerful because it allows for various types of logical, relational, and textual filtering operations. Depending on the type of the query, the results of an SPARQL query are given by SPARQL-specific result sets or by RDF graphs.

2.3.2.3 Enabling Technology Layer

The third layer which is Enabling Technology Layer presents the specifications and technologies to facilitate and enable the development of applications for end-users on the basis of the ontologies defined using the semantics layer. Services are one of the key enabling technologies. Service are described in this section. From the perspective of our research, the broker is also another enabling technology which is discussed in greater detail in section 2.4.

Services

Services are self-contained, reusable, and loosely-coupled distributed software components. As part of the W3C Web Services Architecture Specification [25], an XML-based service description language called WSDL is defined. Services described using WSDL does not provide machine interpretable semantics and this motivated efforts to develop formal descriptions for web services [26]. Service Oriented Architecture (SOA) is an architectural style focusing on building large scale applications using services. In this vein, Semantically Enabled Service Oriented Architectures (SESA) [27] was proposed to develop a comprehensive architecture for integrating the Semantic Web Services (SWS) infrastructures with SOA. SESA effort was targeted towards enhancing and automating service discovery, composition, publishing and monitoring [28]. Services based on SESA are described using semantic-oriented description frameworks.

The semantic-oriented description framework is made up of a description language and associated ontologies. Several standard bodies such as W3C and Semantic Technology Institute (STI) International have published semantic-oriented description frameworks such as W3C Semantically Annotated Web Service Description Language (SA-WSDL) [29], Web Ontology Language for Services (OWL-S) [30], and Web Service Modelling Framework (WSMF) [31].

The main characteristics of the semantic-oriented description frameworks [32] are as follows:

- **Informational/Data Semantics** define input and output messages involved for the technical service interfaces.
- **Functional Semantics** define the service's capabilities using a collection of service interfaces, which can be invoked.
- **Non-Functional Semantics** define non-functional aspects related to the implementation and execution of services (e.g. constraints such as capacity, availability).
- **Technical Semantics** define the supported technical interface protocols, service end-points, and supported serialization formats.

2.3.2.4 Environment Layer

The environment layer enables the execution of semantic web-based applications by providing the essential environment and infrastructure required for execution. It also provides methods to ensure standards and quality expectations for the key enabling layer technology such as services. Environment Layer also takes into account the operating environment for these semantic web-based applications and provides interoperability between different domains. Some of the key technologies and specifications in this layer are application integration and standardization.

2.4 Broker

In this section we give the definition of a broker and explain its key functionalities. Next, we introduce the semantic-oriented broker. Later, we discuss the typical service selection and ranking mechanism for a broker. Finally, we introduce a popular technique for decision-making algorithm used for ranking of services called Multi-Criteria Decision-Making.

2.4.1 Definition and Key Functionalities of a Broker

A broker is an entity that enables the interaction between service clients and service providers by supporting publication, discovery, and binding mechanisms. Publication operations involve service registration and storage of the services. These operations are performed by the service providers in order to advertise their service capabilities to the broker. Discovery operations involve finding the most appropriate service (candidate services) based on the criteria provided by the service client. Binding operations are analogous to invocation operations, which involve the direct interaction between the service client and service provider.

2.4.2 Semantic-oriented Broker

Traditional brokers often lack the expressivity for the stored service descriptions given by the service provider at the time of publishing. To overcome this challenge, semantic-oriented brokers have been proposed to support semantic-oriented description frameworks that contain semantics. This enables efficient service selection and ranking.

2.4.3 Typical Service Selection and Ranking Mechanism for a Broker

The service selection involves the filtering of services based on some concrete criteria. The filtering algorithm for generating a list of candidate services depends on the criteria provided by the client. Service ranking involves the generation of an ordered list of candidate services based on non-functional characteristics (e.g. capacity, availability, performance, cost) of services. This is especially important when the service selection returns several results. In such a case, the service ranking helps to sort the service results based on additional characteristics which are important to the service client (usually based on previous agreements). In a more advanced scenario, the selected services may even be dynamically ranked based on the environmental aspects of the discovery request (service client's geo-location, distance to the service location, reputation of service using previous ratings from the other clients, etc.).

2.4.4 Multi-Criteria Decision-making (MCDM) for Ranking of Services

There are several Multi-Criteria Decision Making (MCDM) algorithms available for making decisions when there are multiple conflicting criteria with varying weightages. These weightages are assigned based on the service client's preferences. Based on the weightages assigned to non-functional characteristics of the candidate services, a score is calculated. The score could be, for instance, a result of pair-wise comparison or an aggregate score indicating a ranking value for a service. The score calculation takes into consideration the type of the non-functional characteristic being evaluated and the MCDM type used. The non-functional characteristics can have either positive tendency (such as availability, security) or a negative tendency (response time, latency or delay). MCDM analysis involves three key steps, [33]

1. Determining the relevant criteria and alternatives
2. Attaching nominal values of relative importance for the criteria and to the impacts of alternatives
3. Processing of the nominal values to determine the ranking value

Following are a list of the popular MCDM algorithms with respect to web service ranking,

- **Weighted Sum Model (WSM) [34] –**WSM solves single dimensional problems, where we have a set of alternatives and a corresponding set of criteria for each alternative. It is one of the oldest MCDM algorithms. The ranking value for each alternative is calculated as sum of the product of the value of the criteria and the nominal value (of importance) of the criteria. The alternative with the highest ranking value is seen as the best choice.

$$A^*_{wsm} = Max \sum_{i}^{j} a_{ij} w_j \qquad \textbf{[34]}$$

- **Weighted Product Model (WPM) [34] –** Another similar method to that of WSM is WPM, where instead of the sum, the product of the ratio of the criteria is calculated and raised to the power of the relative nominal value of the corresponding criterion. Being ratio-based, we make pair-wise comparisons of various alternatives. If the ratio is greater than one, then alternative corresponding to the numerator is ranked higher than alternative corresponding to the denominator. Hence, here we make pair-wise comparisons of the alternatives.

$$R\left(\frac{A_k}{A_l}\right) = \prod_{j=1}^{n}\left(\frac{a_{kj}}{a_{lj}}\right)^{w_j} \quad \textbf{[34]}$$

- **Analytical Hierarchical Process (AHP) –** AHP is another popularly used MCDM algorithm for classifying several unstructured alternatives in hierarchical model and making pair-wise comparisons at each level of the hierarchy. Hierarchy is ordered in such a manner that the goal of making a decision (such as getting a specific type of service) is the root of the hierarchy. The intermediate level consists of a set of criteria. Finally, the alternatives are the bottom of the hierarchy. In AHP, the pair-wise comparisons are made using a standard preference table (e.g. “Equally preferred”, “Extremely preferred”, etc.)

  AHP algorithm involves four steps in order for the decision making process **[34]**,

  1. Structuring of several alternatives into a hierarchical model
  2. Calculation of the weights for each criteria

3. Calculation of the score of each alternative for each criteria based on the weights assigned
4. Calculation of the overall score for each alternative

2.5 Chapter Summary

In this chapter we discussed the background concepts that are related to this thesis. First, we introduced the concept of cloud computing, its benefits and its key facets. Later, we presented conferencing, its key technical components and typical types of conference applications. Then, we discussed the proposed cloud-based conferencing business model.

We followed by introducing Semantic Web, the Semantic Web Layers, and also the specifications and technologies that are relevant to our research. Finally, we discussed the broker as a critical enabling technology for the publication and discovery of services. In this section, we also discussed the relevant sub-tasks for the discovery of services such as service selection and ranking techniques.

Chapter 3

3 Scenarios, Requirements and State of the Art Evaluation

This chapter encompasses four sections. First, we present a set of scenarios that illustrate the use of publication and discovery of substrates in cloud-based conferencing. These scenarios are based on the Cloud-based conferencing business model [3] presented in the previous chapter. Second, we derive a set of requirements for a semantic-oriented description framework and broker architecture for the publication and discovery based on the scenarios. Our proposed semantic-oriented description framework is used to describe the technical and business aspects of cloud-based conference substrates. This semantic-oriented description framework consists of two components: cloud conference ontology and description language. Third, we review the state of the art and evaluate it based on our requirements. Finally, we summarize the chapter.

3.1 Scenarios for Publication and Discovery of Cloud-based Conferencing Substrates

In this sub-section we present three separate scenarios based on the interacting roles in the cloud-based conferencing business model. To provide clarity for the scenarios, we assume three levels of brokers for supporting publication and discovery, based on the interacting roles in the cloud-based conferencing business model. The three levels of brokers are given by level 1 (between conference infrastructure provider and conference substrate provider), level 2 (between conference platform provider and conference infrastructure provider), and level 3 broker (between end users and conference service providers). The first scenario illustrates the interaction between the conference substrate provider, conference infrastructure provider and the level 1 broker. The second scenario illustrates the interaction between the conference infrastructure provider, conference platform provider, conference service provider, and the level 2 broker. The third scenario illustrates the interaction between the conference service provider, conference service end-user and the level 3 broker. The three scenarios form an end-to-end scenario illustrating the flow of publication and discovery interactions from the conference substrate provider to the conference end-user.

3.1.1 Interaction between the Conference Substrate Provider, Conference Infrastructure Provider, and Level 1 Broker

As a first scenario, let us assume that we have two conference substrate providers (CSP1 and CSP2) providing dial-out signaling substrate (S1) with a signaling capacity of 150 users and an audio mixer substrate (S2) with an audio bitrate profile of 128 kbps, latency of 1000 ms, and mixing capacity of 160 users respectively. Both substrate providers publish their substrate offerings to the level 1 broker. The conference infrastructure provider may now issue a request with the following search criteria:

$$(\text{Substrate provider} = \text{CSP1} \cup \text{Substrate provider} = \text{CSP2}) \cap (\text{maximum substrate capacity} \geq 150) \cap (\text{type} = \text{DialOut Signaling} \cup \text{type} = \text{AudioMixer})$$

Note: Search criteria are denoted using basic algebraic notations for clarity of expression.

By issuing such a request to the broker, the conference infrastructure provider discovers the dial-out signaling and audio mixer substrates published by CSP1 and CSP2 respectively.

3.1.2 Interaction between the Conference Infrastructure Provider, Conference Platform Provider, Conference Service Provider, and Level 2 Broker

As a second scenario, the conference infrastructure provider publishes the substrates which were discovered from level 1 broker to the level 2 broker. Alternatively, the conference infrastructure provider may also combine these two substrates to create a dial-out audio conference substrate, also termed as a composite substrate. In this case, the conference infrastructure provider may publish the dial-out audio conference substrate to the level 2 broker. For the sake of conciseness and continuity of the scenario, let us assume that the conference infrastructure provider publishes the dial-out signaling and audio mixer substrates separately to the level 2 broker. Since the substrate description (SD) designed for the higher-level broker (the level 2 broker) will hide some information such as the provider information, conference infrastructure provider alters the substrate description before publication to level 2 broker. After transforming the necessary information, the conference infrastructure provider will publish the two substrate descriptions to the level 2 broker. Now assume that a conference service provider wishes to create a dial-out audio conference application with standard quality and a maximum conference user capacity greater than 100. Assume that a typical audio mixer may support multiple audio quality profiles (e.g. low, standard, and high). The conference service provider will now use the conference platform to discover the constituent substrates for the desired conference type, by issuing a request. The conference platform provider will then issue discovery request to the level 2 broker, on behalf of the conference service provider. The search criteria for the request are as follows:

$(\text{maximum substrate capacity } \geq 100) \cap (\text{type} = \text{DialOut Signaling} \cup \text{type} = \text{AudioMixer}) \cap (\text{quality} = \text{Standard})$

Given such a query, the conference platform provider will discover the substrates for dial-out signaling and audio mixing with standard quality and maximum substrate capacity of 100 (each substrate). Upon discovery of the these substrates from the level 2 broker, the conference platform provider will create a dial-out audio conference application for the conference service provider using these substrates. After the creation, one of the outputs is the composite service workflow file which is stored locally in the conference platform and another output is the description of the composite conference service returned to conference service provider.

3.1.3 Interaction between the Conference Service Provider, Conference End-User, and the Level 3 Broker

In the third scenario, after the creation of the conference service explained in the previous scenario, the conference service provider will publish the description of the conference service to the level 3 broker. Later, the conference end-users who wish to use a dial-out audio conference application discover this conference service description from this broker. The conference end-user may choose to view this conference service description using a browser since the conference end-user can be either an application or a system user.

3.2 Requirements

This section contains the requirements for the semantic-oriented description framework and the broker architecture. The first sub-section outlines the requirements pertaining to the semantic-oriented description framework. The next sub-section lays out the requirements of the broker architecture for publication and discovery in cloud-based conferencing. We draw out these requirements in two sub-sections based on the cloud-based conferencing business model.

3.2.1 Requirements of the Semantic-Oriented Description Framework

In this sub-section, first we derive the requirements related to the semantic-oriented description framework in general and later we derive the requirements specific to the two components of the semantic-oriented description framework. The semantic-oriented description framework entails the following requirements for describing fine-grained conference substrates and composite conference substrates. First, the description framework should be standard-based in order to enable easy interoperability and reuse of existing standard tools. It will also allow easier compliance and will lower the barrier to entry for the providers.

Second, it should support both machine-readable and human-readable representations to enable publication and discovery at all three levels of brokers. For instance, discovery at the level 1 and level 2 brokers are performed by machines. Whereas, the discovery requests at the level 3 broker are performed by end-users or applications, and hence we require human-readability in addition to the machine readability.

Third, the description framework should hide the heterogeneity of the conference substrates/composite conference substrates and provide the substrate interfaces in a uniform manner to facilitate easy interoperability.

Fourth, the description language and cloud conference ontology should accommodate both technical and business aspects. This will allow the infrastructure providers, platform providers, and end users to effectively discover the conference substrates accurately.

Fifth, the chosen description language should be flexible to support a wide range of data formats at substrate interfaces in order to accommodate the needs of different providers and requesters.

3.2.2 Requirements of the Broker Architecture

There are mainly six requirements that should be met in order to enable the usage of the broker by providers and requesters. First, the broker interface for publication and discovery should be independent of the substrates that get stored in the broker. This will make the broker accessible via a unified interface, while hiding the heterogeneity of the conference substrates.

Second, the interfaces should be based on existing standard technologies (protocols/APIs) to enable easy interoperability and reuse of existing standard tools.

Third, the architecture should support easy interoperability. The interface should be flexible in terms of the supported serialization formats for description. This allows the providers to publish and retrieve the description of conference substrates and composite conference substrates using several serialization formats such as XML, JavaScript Object Notation (JSON), and plain text. Additionally, it should also support transformation to a human-readable format (such as HTML/XHTML) to enable discovery from broker by end-users using tools such a Web browser.

Fourth, the discovery interface should consider both technical and business aspects. This helps the conference infrastructure providers, conference platform providers, and conference end-users to perform easy and accurate discovery operations on the brokers.

Fifth, the broker should provide an extensible architecture to support substrate description defined using an existing framework chosen or provide an explicit support for new semantic-oriented description frameworks.

Sixth, the broker should be able to select and rank the conference substrate(s) based on aspects such as the constraints defined inside the substrate description. This ensures the discovery of the most appropriate results based on several criteria (e.g. cost, latency, capacity).

3.3 The State of the Art Review

In this section, we organize the state of the art into two categories. In the first sub-section we discuss the state of the art for semantic-oriented description framework and in the second sub-section we discuss the state of the art for the broker architecture for the publication and discovery of substrates in cloud-based conferencing.

3.3.1 Semantic-oriented Description Framework

Standardization organizations such as W3C, IETF, and STI are involved in the development and maintenance of proposals for semantic-oriented description frameworks. The most prominent initiatives for semantic-oriented description frameworks include Web Ontology Language for Services (OWL-S) [30], Web Service Modelling Ontology (WSMO) [35], WSMO-Lite [32], SAWSDL [29], and Linked Unified Service Description Language (Linked USDL) [36]. In the following sub-sections, we start by discussing each of these semantic-oriented description frameworks and their associated components: ontology and language. Later, we discuss the components of each semantic-oriented description framework separately. Finally, we evaluate the state of art for semantic-oriented description framework and its constituent components in the light of the requirements stated in section 3.2.1.

3.3.1.1 Semantic Web Ontology Language (OWL-S)

W3C OWL-S (formerly DAML-S[37]) is a semantic-oriented description framework for services. The framework consists a set of three key sub-ontologies for describing service concepts and OWL [38] as a description language. The three key sub-ontologies are defined as follows [38]:

- **Service Profile** provides information about the high-level functional capabilities of the service, service category, input-output-preconditions-effects (IOPE), and provider information. The service profile also describes non-functional properties of the service (e.g. cost, quality). This information is used for the discovery of the service.

- **Service Model** provides information about how to use a service. It views and models a service as a process using standards in process modeling and workflow technology such as NIST's Process Specification Language (PSL) and Workflow Management Coalition Effort[39].
- **Service Grounding** provides information about how to access service end-points (i.e. data formats, protocols supported by the service end-points defined in service model). The term "service grounding" is commonly utilized in Semantic Web vocabulary to indicate the technical aspects of the semantic-oriented services (such as service end-points, interfaces, input, and output). OWL-S does not describe a specific service grounding mechanism to be used. However, only a Web Service Description Language (WSDL)-based ground mechanism is specified as part of the original specification [40] as illustrated in Figure 7. WSDL contains the technical aspects of the service (i.e. input, output, fault types, service interfaces, protocol bindings, and service end-points).

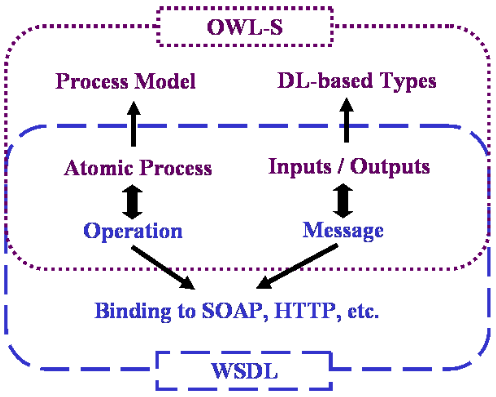


Figure 7: WSDL-based Service Grounding for OWL-S [30]

3.3.1.2 Web Service Modeling Ontology (WSMO)

Web Service Modeling Ontology (WSMO) [35] is an initiative by the STI WSMO Working Group with the main goal of providing a conceptual model for semantic-oriented services. Later in 2005, it was accepted as a W3C submission. WSMO consists of ontologies for describing Web services (based upon Web Service Modeling Framework [31]) and the Web Service Modeling Language (WSML) as the description language. WSMO provides ontologies for four core elements. The core elements are as follows [31]:

- **Domain Ontologies** represent the concepts and relations, which are relevant to the specific domain of the target service implementation (e.g. telecom, retail, biomedical).

- **Web Services** represent technical aspects such as interfaces, inputs, and outputs for the Web service. It also describes some behavioral capabilities of services in terms preconditions, assumptions, post conditions, and effects.
- **Goals** represent the outcome desired by the service requester upon the successful execution of the service (e.g. participant getting connected to a specific conference session).
- **Mediators** represent a set of concepts that help to provide inter-operability between the core elements.

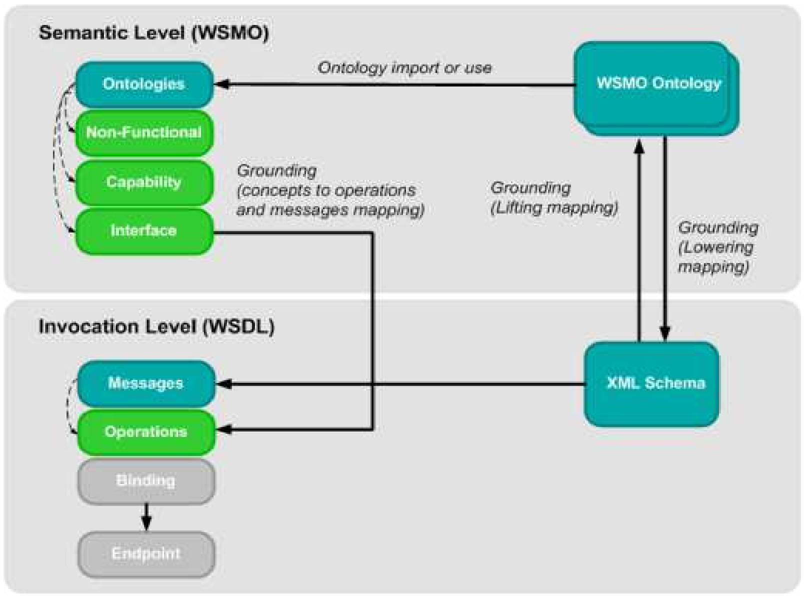


Figure 8: Semantic Service Anatomy [41]

The language choice WSML consists of several variants depending on expressivity of the statements, much similar to OWL variants. WSML is based on knowledge representation techniques such as description logic, first-order logic and logic programming. Similar to OWL-S, in WSML the technical realization of services is described using WSDL files as illustrated in Figure 8 above. Though WSMO is independent from the service grounding description's serialization format, only WSDL-based service grounding is officially specified.

3.3.1.3 Linked USDL

Linked USDL [36] was proposed by the W3C USDL Incubator Group, which is a joint effort between SAP Research Labs and Knowledge Media Institute of The Open University. Linked USDL draws its specifications based on W3C Unified Service Description Language [42]. Linked USDL consists of a set of three key sub-ontologies: USDL-Core, USDL-Pricing, and USDL-SLA. The sub-ontologies reuse standardized or widely accepted upper ontologies such as

Friend of a Friend (FOAF) [43], Dublin Core [44], and Good Relations [45]. Linked USDL uses OWL as description language choice.

The following are the three key sub-ontologies of Linked USDL:

- **USDL-Core** represents the concepts and relationships for modeling the technical aspects of the services. Services are modeled as resources to indicate a concrete object with underlying implementations. These resources provide composability of services. Composability of services allows complex services to be composed of multiple atomic services with fine-grained functionalities. Technical aspects are represented using the Minimal Service Model (MSM) vocabulary [46] rather than WSDL. MSM is a simple and light-weight RDF-based service description meta-model designed account for the essential aspects of service invocation like service end-points, operations, inputs, outputs, and exception handling.
- **USDL-Pricing** represents a pricing model for services based on the GoodRelations ontology.
- **USDL-SLA** represents vocabulary used to specify qualitative aspects and agreements between the service provider and the service requestor. The Service Level Agreement (SLA) represents a contractual agreement between the service provider and service requester. It provides specific conditions under which the services are expected to be delivered.

3.3.1.4 WSMO-Lite

WSMO-Lite [32] is a light-weight semantic-oriented description framework that draws its specifications closely from its parent - WSMO Framework. WSMO-Lite provides a minimal ontology to describe the service's classification (i.e. service category), and non-functional parameters (such as quality of service or policies). WSMO-Lite provides only the relevant ontologies which are agnostic of the underlying language component (Figure 9). As part of the specification, it consists of WSDL or non-standard hRESTS as service description language [47].

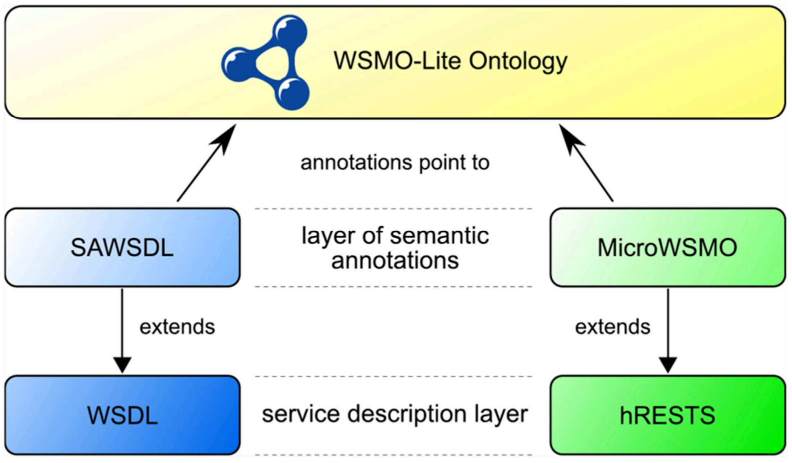


Figure 9: WSMO-Lite Semantic Layering [32]

The mapping between WSDL service description and WSMO-Lite is enabled by using Semantic Annotations for WSDL and XML Schema (SAWSDL) [29] as illustrated in Figure 10. SAWSDL is a W3C-recommended XML schema specification for annotating WSDL-based services. SAWSDL XML schema specification extends the original W3C WSDL specification with attributes such as *modelReferences* that help to attach semantic concepts and relations to the WSDL. The mapping between hRESTS and WSMO-Lite is enabled by using MicroWSMO [48].

MicroWSMO is a non-standard annotation mechanism to enable service annotation for representation formats such as HTML/XHTML.

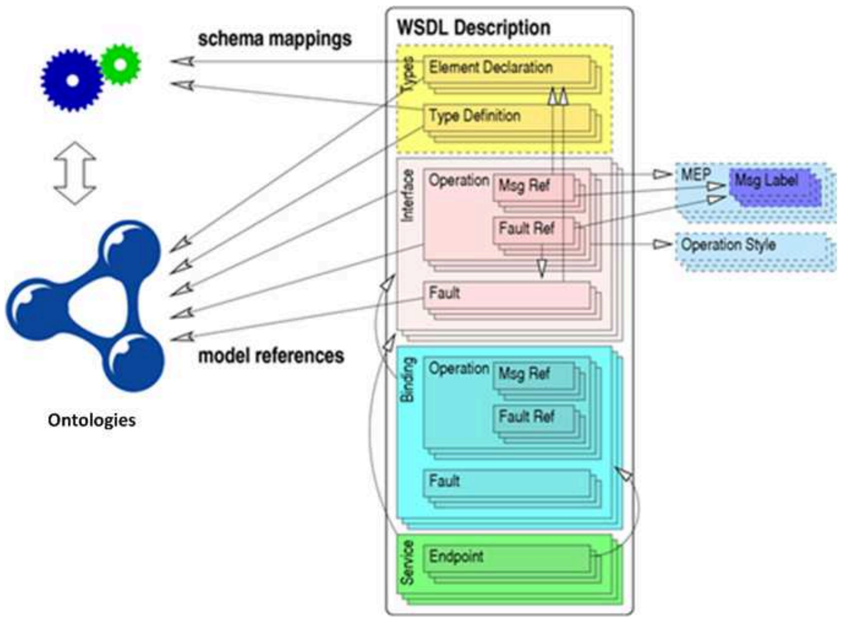


Figure 10: SAWSDL Mapping between WSDL and Ontologies [29]

3.3.1.5 Components of the Semantic-oriented Description Framework

In this sub-sub-section, we discuss the state of the art related to the components of the semantic-oriented description framework. We start by discussing the related description languages. Later, we discuss the related ontologies for cloud computing and conferencing.

3.3.1.5.1 Related Description Languages

Besides the semantic-oriented description frameworks, it is worth reviewing the existing service description languages. Machine-readable service description languages, such as W3C WSDL, provide mechanisms for describing the technical aspects of both SOAP-based and HTTP-based services (e.g. RESTful Web services). In contrast, W3C WADL supports the description of RESTful Web services alone. They are both syntactic-oriented approaches and do not offer a mechanism for describing the semantics.

3.3.1.5.2 Related Ontologies for Cloud Computing and Conferencing

Some ontologies related to cloud computing and conferencing have been proposed which are relevant to our research. For example, in reference [49], the authors propose ontologies for describing services in converged telecom networks. IETF has proposed an XML-based conference information data model for centralized conferencing [9]. It provides a mechanism to specify fundamental aspects of a centralized conference such as conference description, participant's information, and conference state. In reference [50], Aakif et al. proposed a telecommunication ontology by extending WSMO-based ontologies. These approaches fail to describe the business aspects such as provider information, constraints, and price model (e.g. based on volume, number of users, etc.).

There are other ontologies relating specifically to cloud computing such as ontologies described in references [51], [52], and [53]. Reference [51] proposes a mediation ontology to specify infrastructure-level resources of the disparate cloud providers. Reference [52] proposes a similar mediation ontology for the platform-level concepts such as programming language, software framework, and application software security. Mediation ontology helps to provide standard interoperability between disparate systems or providers. Reference [53] proposes an ontology for describing the constraints for cloud-based services such as performance, availability, and efficiency. However, none of them have ontologies to describe cloud-based telecommunication services such as conferencing.

3.3.1.6 Evaluation of the State of the Art in the Light of the Requirements

We have done a review of prominent semantic-oriented description frameworks such OWL-S, WSMO, Linked USDL, and WSMO-Lite. Furthermore, we have also reviewed the constituent description languages, and also the relevant ontologies for cloud computing and conferencing separately. As far as the first requirement (which is a standard-based semantic description framework) is concerned, the description frameworks discussed are accepted as a W3C submission.

The second requirement dictates the support for both human and machine readability. Apart from WSMO-Lite, the other description frameworks fail to explore the possibility of human-readable description such as hRESTS. The primary reason for this is that the main goal for the development of frameworks such as WSMO-Lite is to serve an annotation mechanism to existing languages such as HTML, whereas the main goal of the other semantic-oriented description frameworks discussed is to provide a standalone machine-readable service description mechanism.

From the perspective of the third requirement, reviewed semantic-oriented description frameworks could support uniform interfaces using technologies like RESTful web services.

For the fourth requirement, to the best of our knowledge, the reviewed semantic-oriented description frameworks do not have specific ontologies to describe the business and technical aspects of cloud-based conference substrates. OWL-S and WSMO do not use well-established ontologies to describe business aspects and choose to define their own set of ontologies. WSMO-Lite does not provide ontologies to support or describe business aspects and hence it depends on developers to create their own ontologies. Linked USDL framework exploits the use of well-established ontologies such as GoodRelations to describe the business aspects (e.g. cost, constraints). However, even Linked USDL does not provide specific ontologies to describe business and technical aspects of cloud-based conferencing specifics. Besides none of the reviewed semantic-oriented description frameworks provide support for conference specific technical aspects such as asynchronous communication (e.g. Prompt and Switch feature for dial-in conferencing [54]).

From the perspective of the fifth requirement, none of the reviewed semantic-oriented description frameworks discuss a specific vocabulary to define multiple data formats for the service interfaces.

After considering all the limitations of the existing semantic-oriented description frameworks explained above, we conclude that they do not fully satisfy all the requirements (stated in the section 3.2.1) for describing the substrates in cloud-based conferencing.

In the following Table 1, we summarize our evaluation of the semantic-oriented description frameworks.

Requirements \ Frameworks	OWL-S	WSMO	Linked USDL	WSMO Lite
Standards-based	Satisfied	Satisfied	Satisfied	Satisfied
Machine and human readability	Not discussed	Not discussed	Not discussed	Satisfied
Reduce the heterogenity of substrates	Satisfied	Satisfied	Satisfied	Satisfied
Accommodate the technical and business aspects of the conference substrates	Not Applicable	Not Applicable	Partially Satisfied	Not Satisfied
Support for multiple data formats for service interfaces	Not Satisfied	Not Satisfied	Not discussed	Not Satisfied

Table 1: Summary of Evaluation of Semantic-Oriented Description Frameworks

3.3.2 Broker Architecture for Publication and Discovery of Substrates in Cloud-based Conferencing

Several broker architectures have been proposed. The most prominent broker architectures are: Universal Description, Discovery and Integration [55], FUSION Semantic Registry [56], Semantic Repository for Adaptive Services [57], Cloud Infrastructure Service Broker [53], Web Service Modelling eXecution environment (WSMX)-based Broker [50], AtomServ [58], and iServe [46]. In the following sub-sections, we discuss each of these broker architectures. Finally, we evaluate the state of the art for broker architectures in the light of the requirements stated in section 3.2.2.

3.3.2.1 Universal Description, Discovery and Integration (UDDI)

The best-known broker architecture for publication and discovery of web services is Universal Description, Discovery and Integration [55] (UDDI). UDDI is an XML-based registry with a well-defined structure for defining technical aspects (described using WSDL) and business aspects (as part of the UDDI schema). It provides standard interfaces for publication and discovery using publishing and inquiry APIs, respectively. The discovery operations for service descriptions can be done using a text-based search using the keywords contained in service names [56].

3.3.2.2 FUSION Semantic Registry

The FUSION Semantic Registry project [56] is a reference architecture with the goal of semantically augmenting the UDDI architecture and therefore providing an accurate discovery mechanism. FUSION semantic registry consists of a publication manager for managing publication from service providers based on SAWSDL based descriptions. Besides, it consists of a discovery manager with OWL ontology processing and OWL-DL reasoning capabilities for automated discovery [56].

3.3.2.3 Semantic Repository for Adaptive Services

Semantic Repository for Adaptive Services [57], proposes a semantic repository supporting service description languages such as WSDL and composition description languages such as BPEL, with semantic-oriented indexes to describe the non-functional properties (e.g price, capacity) for the descriptions. Since the indexes are maintained separately, there is clear distinction of semantic-oriented information and implementation information described using description languages (e.g. WSDL and BPEL). Due to this distinction, the proposed architecture can support additional description languages. It also proposes a query language *OntoQL* for discovery of services by exploiting the semantic-oriented indexes [57].

3.3.2.4 Cloud Infrastructure Services Broker

The Cloud Service Measurement Index Consortium (CSMIC) [53] proposes a cloud-based broker, which allows classifying of services based on the *Cloud Service Measurement Index*. Cloud Service Measurement Index is a set of key constraints for cloud infrastructure services (i.e. services provided by the IaaS). The proposed constraint definitions are based an elaborate QoS (Quality of Service) Ontology [59] for web services, which have been proposed earlier. The broker architecture allows ranking of services based on set of infrastructure-level constraints (such as disk quota, main memory, processors). Ranking and selection of services is done using Analytical Hierarchical Process (AHP) [60], which is a type of Multi Criteria Decision Making (MCDM) technique.

3.3.2.5 Web Service Modelling eXecution environment (WSMX)-based Broker

In reference [50], Aakif et al. proposes a WSMX-based semantic-oriented broker architecture for automated discovery and execution of telecom-based semantic web services (for billing applications). WSMX [61] is the reference implementation for automating publication and discovery using WSMO. As part of the reference, a semantic-oriented framework is proposed to support automated discovery of heterogeneous and homogeneous services in the telecommunication industry. The WSMX architecture follows a goal-based discovery for services. The goal- based descriptions are originally specified using a WSML [62] and later converted to a native XML-based format. This XML-based format is sent as content of a SOAP-based discovery request [63].

3.3.2.6 AtomServ – Atom-based Service Discovery Architecture

AtomServ [58] proposes a service discovery architecture based on IETF Atom syndication format (RFC 4287 [9]) and Atom Publication/Subscription Protocol [64]. It uses internal transformation tools to convert WSDL files to Atom feeds. One of the key goals of the broker architecture is usability. AtomServ's usability is improved by providing a simple service discovery mechanism for end users. For example, the end users are able to find, subscribe, and invoke web services using just the widely available tools such as a browser. AtomServ supports two types of discovery mechanisms: keyword-based and concept-based (using semantic-oriented approach) mechanisms. Out of the two discovery mechanisms, only keyword-based discovery is discussed in detail as part of the reference.

3.3.2.7 iServe Semantic-oriented Broker Architecture

iServe [46] provides a novel and extensible broker architecture that allows the publication and discovery with support for multiple semantic-oriented descriptions. It provides simple, uniform interfaces (via RESTful APIs) with flexibility of serialization formats for the semantic-oriented service descriptions and an accurate discovery mechanism using the W3C SPARQL specification. iServe has a transformation engine that converts all the published semantic-oriented services into Minimal Service Model (MSM) service ontology.

3.3.2.8 Evaluation of the State of the Art in the Light of the Requirements

We have done an extensive review of most prominent broker architectures such as UDDI, FUSION Semantic Registry, Semantic Repository for Adaptive Services, Cloud Infrastructure

Service Broker, Web Service Modelling eXecution environment-based Broker, AtomServ Broker, and iServe architecture. As far as the first requirement is concerned, all the reviewed broker architectures except Cloud Infrastructure Service Broker provide interfaces independent of the stored conference substrates. Cloud Infrastructure Service Broker does not discuss the interfaces for publication and discovery.

For the second requirement, the UDDI, FUSION semantic registry, and WSMX-based broker build publication and discovery interfaces on SOAP-based APIs. As stated earlier, Cloud Infrastructure Services Broker does not discuss the details of how the interfaces for publication and discovery are implemented. AtomServ utilizes the Atom Publishing Protocol [58] to support both publication and discovery using standard HTTP-based operations, which are exposed as REST-based API. iServe supports publication and discovery using REST-based API.

For the third requirement regarding serialization formats, UDDI and FUSION semantic registry support only XML-based serialization formats. The AtomServ architecture is strictly based on Atom feeds which are restricted to XML-based serialization format Cloud Infrastructure Services broker and WMSX-based broker, does not discuss specific serialization formats supported. iServe is the only architecture capable of supporting for multiple serialization formats (e.g. XML, Turtle, N3 [25], and JSON [65]) to the best of our knowledge.

From the perspective of fourth requirement, it is necessary to specify both business and technical aspects using standard technologies. The existing query/discovery interface in UDDI is non-expressive, because of the absence of machine-processable semantics [66] [67]. This inhibits accuracy of the discovery operations [56]. FUSION semantic registry [56] aims to provide better expressivity than UDDI by including semantic information about the services. However, the reference [56] does not discuss the details of discovery query specification using any standard discovery query specification technology. In Semantic Repository for Adaptive Services, the discovery mechanism uses a non-standard specification language called OntoQL to specify business and technical aspects of the services. WSMX uses WSML (W3C Submission [68]) to specify both business and technical aspects. AtomServ only discusses keyword-based discovery mechanism which lacks the capability to describe specific business and technical aspects as part of a discovery request (e.g. cost per month less than 10 dollars). On the other hand, iServe allows standards-based powerful query specification using W3C SPARQL [24] in order to describe both business and technical aspects in a concise manner.

From the perspective of the fifth requirement, UDDI, FUSION semantic registry, and WSMX-based broker focus on providing support for specific service descriptions (e.g. WSDL, SAWSDL, and WSMO). Cloud Infrastructure Services broker does not discuss supported service descriptions. Semantic repository for adaptive services provides an extensible architecture due to the separation of semantic-oriented indexes and the service description. iServe also provides an extensible architecture which is capable of supporting additional service descriptions.

From the perspective of sixth requirement, regarding ranking and selection mechanism, UDDI, FUSION Semantic Registry, Semantic Repository for Adaptive Services, WSMX-based broker, AtomServ, and iServe [46] does not discuss specific ranking algorithms utilized for selection and ranking during discovery. Cloud Infrastructure Service Broker [53] uses AHP-based ranking algorithm to enable efficient discovery of services.

Based on our review of prominent broker architectures, we found that iServe is only broker architecture that satisfies our requirements 1 to 5 (stated in section 3.2.2). However, iServe still does not provide a selection and ranking mechanism based on constraints (e.g. price, capacity, latency), according to our sixth requirement.

In the following Table 2, we summarize our evaluation of the broker architectures.

Broker Architectures / Requirements	UDDI	FUSION *	Repository for Adaptive Services *	Cloud Services Broker	WSMX-based	AtomServ	iServe *
Interfaces should be independent of substrates	Satisfied	Satisfied	Satisfied	Satisfied	Satisfied	Satisfied	Satisfied
Interfaces should be based on existing standard protocols/APIs	Satisfied	Satisfied	Not discussed	Not discussed	Satisfied	Satisfied	Satisfied
Support several serialization formats (JSON, XML, HTML)	Not Satisfied	Not Satisfied	Not discussed	Not discussed	Not discussed	Not Satisfied	Satisfied
Discovery specification with both technical and business aspects (together) using standard technologies	Not Satisfied	Satisfied	Not Satisfied	Not discussed	Satisfied	Not discussed	Not discussed
Support for an extensible architecture	Not Satisfied	Not Satisfied	Satisfied	Not discussed	Not discussed	Not discussed	Satisfied
Selection and Ranking based on conference specific constraints	Not discussed	Not discussed	Not discussed	Satisfied	Not discussed	Not discussed	Not Satisfied

Table 2: Summary of Evaluation of Broker Architectures

3.4 Chapter Summary

In this chapter we presented three scenarios that illustrate the publication and discovery of substrates in cloud-based conferencing. Then, the requirements were drawn based on these specific scenarios and classified into two sub-sections: semantic-oriented description frameworks and broker architecture. The first sub-section outlined the requirements related to the description of conferencing substrates. The second sub-section outlined the requirements for the broker architecture to enable the publication and discovery of substrates in cloud-based conferencing.

Finally, we presented the state of the art for semantic-oriented description frameworks and broker architecture. We also discussed the limitations of reviewed semantic-oriented description frameworks and broker architectures on the basis of the outlined requirements.

Chapter 4

4 Proposed Architecture

This chapter starts by discussing the proposed semantic-oriented description framework in detail. Later, it describes the proposed broker architecture for the publication and discovery of conference substrates. Soon after, it provides an end-to-end publication and discovery steps and an illustrative scenario for publication and discovery of substrates. Finally, we summarize this chapter.

4.1 Proposed Semantic-Oriented Description Framework

The proposed semantic-oriented description framework defines a new cloud-based conferencing ontology and uses OWL as the description language. The cloud-based conferencing ontology consists of three key constituent sub-ontologies: common ontology, Level 1 and Level 2 broker ontology, and Level 3 broker ontology. In this section, we start by providing an overview of the proposed semantic-oriented description framework. Next, we discuss the constituent sub-ontologies of cloud-based conferencing ontology in detail.

4.1.1 Overview of the Proposed Cloud Conferencing Ontology

Among the description frameworks reviewed, we propose reusing Linked USDL [36], which is a good candidate for a semantic-oriented description framework. In addition, we propose the following extensions:

- Creation of cloud-based conferencing ontology to support both business and technical aspects of the cloud-conferencing specifics.
- Using the W3C SA-REST as service ontology instead of MSM, due to its support for light-weight and uniform interfaces of the conference substrates. The existing MSM service ontology (Linked USDL service ontology) fails to address the support for asynchronous communication (e.g. Prompt and Switch feature for dial-in conferencing [54]).

It reuses existing upper ontology concepts (e.g. Dublin Core [44], GoodRelations [45]) and extends them to meet cloud-based conferencing specifics.

4.1.2 Common Ontology

The common ontology illustrated in Figure 11 describes technical aspects of interface that are common across conferencing substrates (e.g. dial-out signaling, dial-out audio mixer substrate). The interfaces are described through the set of operations they encompass, along with the inputs and outputs of each operation. The operations are described using the SA-REST, which we extend as part of our proposed ontology in order to support asynchronous operations. SA-REST provides a light weight service ontology to define the operations, inputs, and outputs. We extended it by adding a collection of seven properties to define an asynchronous callback endpoint and supported data formats for the service interfaces. When calling an asynchronous operation (e.g. inviting a participant to join a conference), the client receives an intermediate response informing that the request is being processed. The intermediate response is sent while the actual operation is not yet completed (e.g. the requested participant has not yet joined the conference). The client uses the operation properties to specify the callback endpoint reference, where to asynchronously notify the requester, and when the operation is actually completed (e.g. target participant has successfully joined the conference). The proposed mechanism includes the callback URI and parameters (e.g. the callback body parameters and URI parameters).

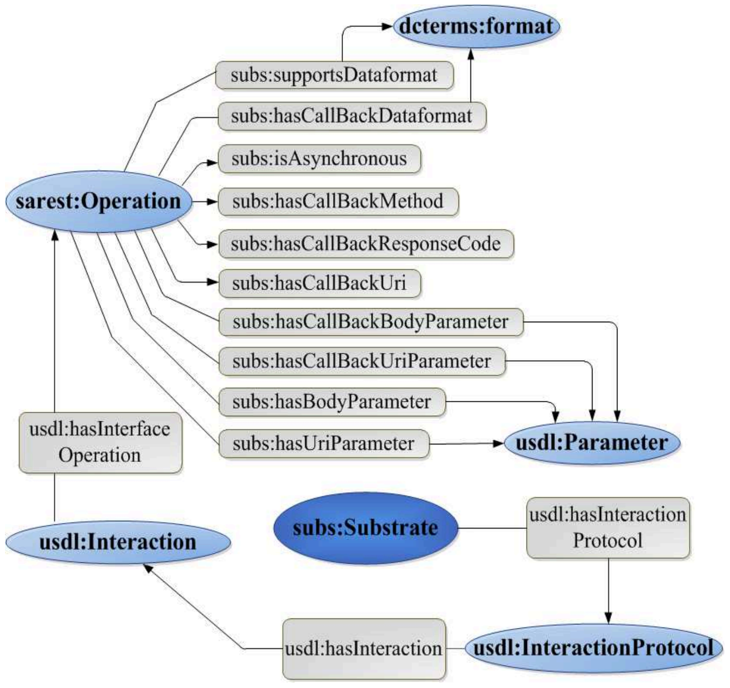


Figure 11: Common Ontology

The operation parameters can be specified as either URI or body parameters, depending on the parameter value's size. Big parameters should be enclosed within the operation message body, whereas short parameters might be appended to the message URI. In addition, we also define the vocabulary to specify the data formats supported by each operation. Hence, each operation may render the response in different data formats. The supported data formats are described using the *supportsDataformat* and *hasCallBackDataformat* properties.

4.1.3 Level 1 and Level 2 Broker ontology

The Level 1 and Level 2 broker ontology in Figure 12 describes the business aspects of the associated providers: substrate provider, infrastructure provider, and platform provider. The lower level provider publishes to the broker (Substrate Provider to level 1 Broker and Infrastructure Provider to level 2 Broker) and a higher layer provider discovers from the broker (Infrastructure Provider from level 1 Broker and Platform Provider from level 2 Broker). Business aspects include information such as the provider's information and subscription information that bind the providers and consumers (i.e. which infrastructure provider is subscribed to which substrate). Providers in the cloud-based conferencing business model [3] provide their substrates in terms of offerings. An offering may include either an atomic substrate or a composite substrate. The offerings are modeled in the ontology using the USDL *Offering* class.

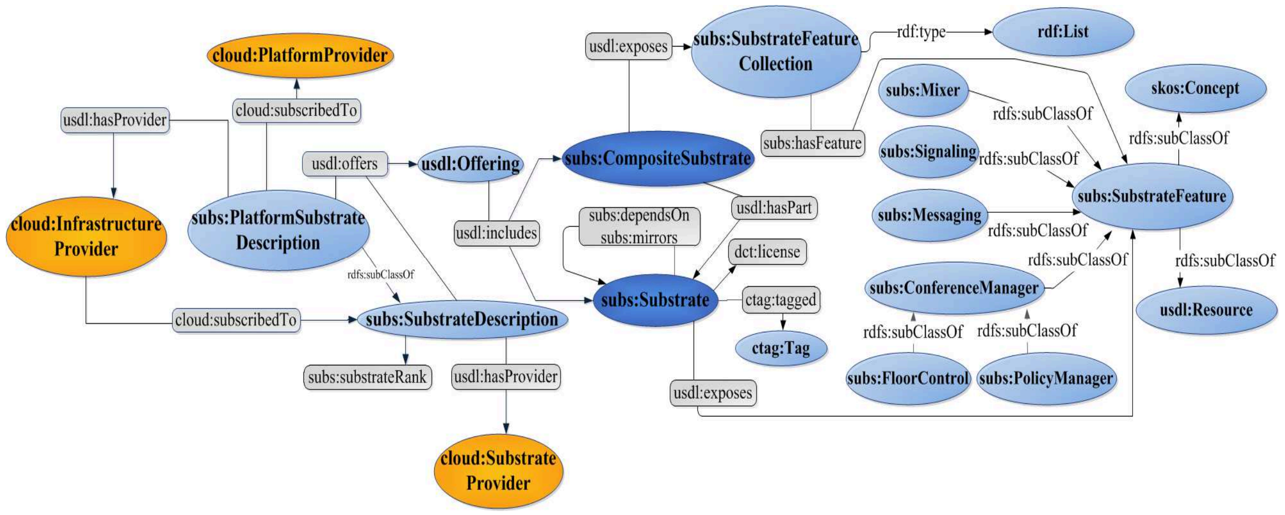


Figure 12: Level 1 and Level 2 Broker Ontology

In the ontology, the substrates are modeled as Linked USDL *Services*, thereby allowing the reuse of the Linked USDL Pricing model to define the pricing (e.g. per user, per month, discounts, etc.). Furthermore, the Linked USDL SLA is reused to describe other constraints information (e.g capacity, availability) apart from pricing. In the ontology, composite substrates are modeled as Linked USDL *CompositeServices*. Such modeling allows constraint specification for atomic substrates and composite substrates. The semantic-oriented description framework is also capable of handling a variety of constraint types such as numeric (e.g. latency), vectors, numeric ranges, boolean values, and string values (e.g. audio/video codecs).

Conference substrate providers who publish to the level 1 broker provide the fine-grained substrates. Conference infrastructure providers who publish to the level 2 broker provide either atomic substrates or composite substrates. Hence, it is necessary to differentiate between them by indicating the type of functional feature(s) (e.g. audio mixing or/and signaling) supported by these substrates. Atomic substrates have a single functional feature (denoted by *SubstrateFeature*) described using the *exposed* property. In contrast to atomic substrates, composite substrates have multiple functional features (denoted by *SubstrateFeatureCollection*) described using the *exposed* property as an RDF *List*. So, *SubstrateFeatureCollection* allows us to define a standard RDF container consisting of multiple functional features of the conference substrate. For simplicity, the ontology (Figure 12) only describes the high-level functional features of conference substrates (such as signaling, mixing, and advanced conference control features such as floor control and policy management). A more detailed look at the substrate's functional features is illustrated in Figure 13.

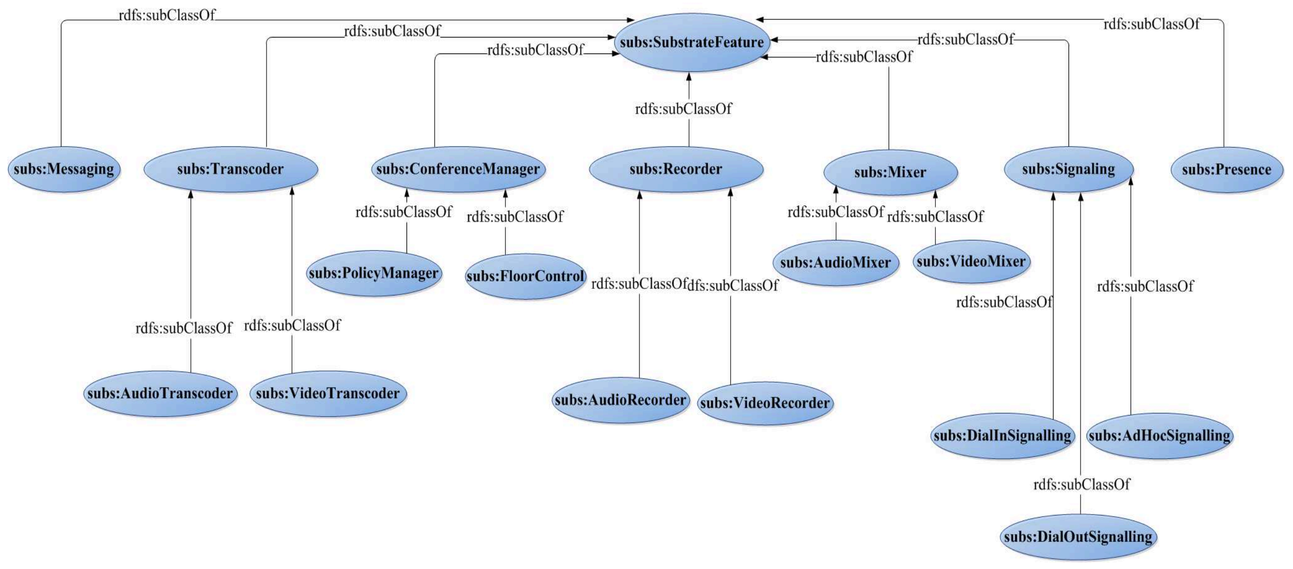


Figure 13: Identified Substrate Features

4.1.4 Level 3 Broker Ontology

The Level 3 broker ontology in Figure 14 describes the business aspects of the associated providers and consumers namely: service provider (*ServiceProvider*) and end user (*ConferenceEndUser*). Since the level 3 broker provides conference services, the ontology provides in-depth information about the conference and its participants. A conference service is depicted as a specific type of composite substrate. A dial-out audio conference, for instance, can be described as a composition of a dial-out signaling and audio mixer substrates. In the ontology, a conference is also defined as a Linked USDL resource to capture the fact that it is the concrete object that implements the conference service.

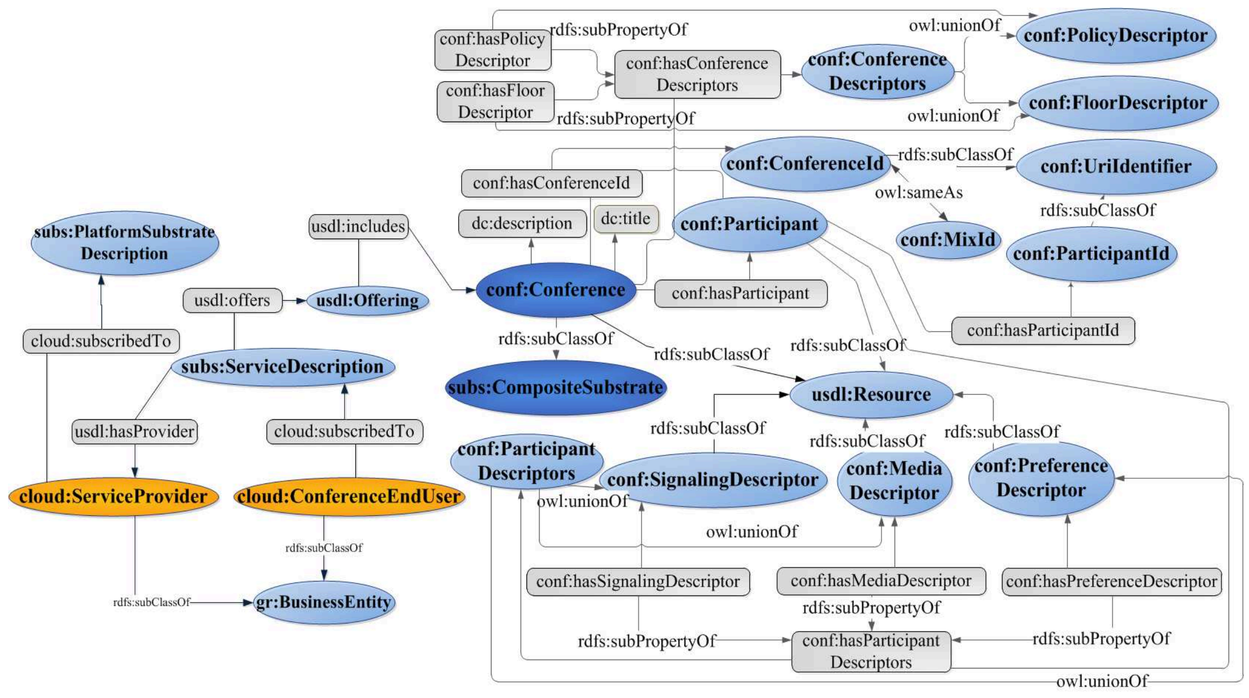


Figure 14: Level 3 Broker Ontology

The participants are described using three important descriptors – signaling, media, and preference descriptors.

- **Signaling descriptor** includes signaling information such as the participant identifier and signaling session description information.
- **Media descriptor** gives the media characteristics of the participant's ongoing media session such as the media transport address, and port number.
- **Preference descriptor** details the participant preferences such as the media codec's priority and presence information of the participant.

To describe the conference substrates in an unambiguous manner, the semantic-oriented description framework uses W3C-recommended OWL as the language choice. This provides the extensibility and serialization format flexibility (because of RDF-based serialization) required for our description framework.

4.2 Proposed Broker Architecture

In this sub-section we will start by providing an overview of the broker architecture. Next, we categorize and discuss the constituent components of the broker architecture in detail.

4.2.1 Overview of the Broker Architecture

Figure 15 illustrates the broker architecture for publication and discovery in cloud-based conferencing. The broker architecture reuses some of the components from iServe architecture [46] viz, publication and discovery interfaces and semantic data store interfaces. The providers and consumers communicate with the broker via a REST API. We chose REST because it offers a uniform interface and it is flexible in terms of the supported serialization formats for description of substrates and data formats in the service interfaces. The discovery requests are described using SPARQL specification, and they are transferred as content of REST requests. We selected SPARQL because it is standard, semantic-oriented, and can be used to express rich and expressive queries across diverse data sources. The broker uses a semantic data store to save the descriptions of substrates. During publication of the description document, the cloud-based conferencing ontology serves as reference ontology for the validation.

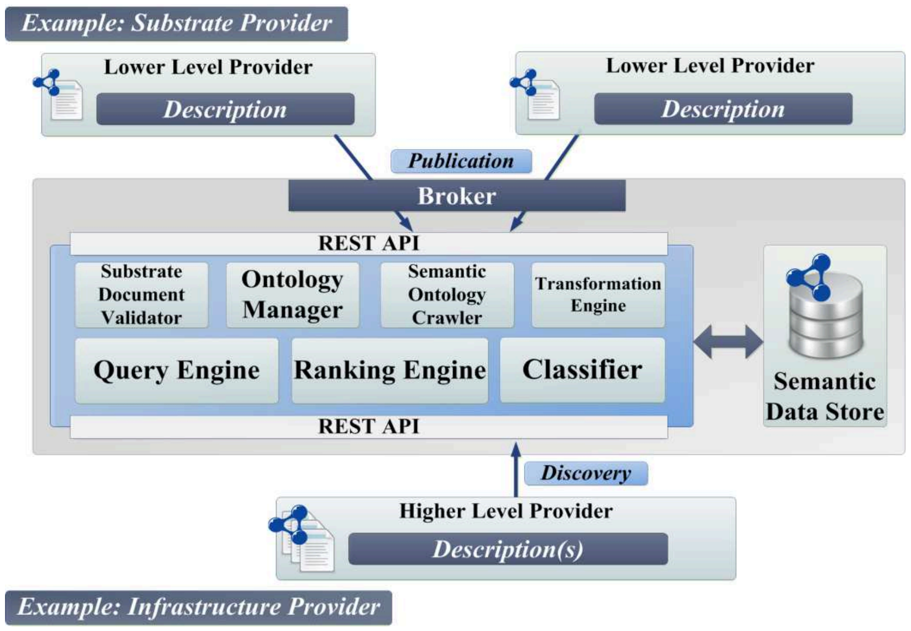


Figure 15: Broker Architecture

4.2.2 Components of the Broker Architecture

In the sub-section, first we classify the different components of the broker into three categories based on their functionality. Later, we discuss each of categories along with their constituent components.

4.2.2.1 Categorization of Constituent Components of the Broker Architecture

The broker includes a set of supporting components to access, validate, and manage the description documents and the cloud-based conferencing ontology. These components can be classified into following three categories,

- The *first* category supports the *validation and the management of the descriptions*, and it includes the document validator and the classifier.

- The *second* category is used for the *management of the cloud-based conferencing ontology* and it consists of the ontology manager and the semantic ontology crawler.
- The *third* category enables *efficient discovery of substrates* and it contains the query and the ranking engines.

The transformation engine is a common supporting component used across all of the categories. The description document(s) for the selected result(s) is reformatted – if needed – according to the data format (e.g. XML, JSON, HTML+RDFa) supported by the requester. The level 1 and level 2 brokers require a machine interpretable description language, whereas the level 3 broker requires a human-readable description language for discovery by end-users. This transformation from/to a machine-readable format (e.g. JSON, XML) or a human-readable format (e.g. HTML+RDFa) is performed by the transformation engine.

4.2.2.2 Validation and the Management of the Descriptions

Descriptions are validated by the substrate document validator and managed by the classifier. These two components are explained in detail in the following sub-sections.

4.2.2.2.1 Substrate Document Validator

The substrate provider, infrastructure provider, or service provider may choose to publish the description document in any of the supported RDF serialization formats. Prior to storing a published document, the broker converts the document into XML format using the transformation engine. Then it checks the document validity against the cloud-based conferencing ontology and the external ontologies (such as Dublin Core[44] and

GoodRelations[69]). This function is handled by the substrate document validator, which seeks the help from ontology manager to retrieve the latest version of the relevant ontologies from the semantic data store based on the ontology references.

Once the validation is completed, the description document is stored in the semantic data store. Each description document is assigned a Globally Unique Identifier (GUID) that allows the broker to create a description under a named graph. Named graphs are addressed by a context URI and it allows the broker components (e.g. query engine) to retrieve the description documents from semantic data store in an easier manner. Algorithm 1 listed below illustrates the basic steps performed during the publication of a description document.

```
Algorithm 1 PublishDescriptionDocument(Description d)
$O^{1..3}$: Three key sub-ontologies of the Cloud Conference Ontology
$O_e^{1..n}$: The n external ontologies such as Dublin Core, GoodRelations
1: Check document serialization format
2: Transform $d$ to RDF/XML syntax, if required.
3: Parse and Validate() the description $d$ against the sub-ontologies $O^{1..3}$ and the ontologies $O_e^{1..n}$
4: Generate $n^d$ indicating the unique named graph by assigning contextual URI for given description
5: Add the description identified by the $n^d$ into the repository.
```

4.2.2.2.2 Classifier

The classifier performs indexing of the conference substrates based on certain properties in order to improve the performance and efficiency of the discovery operation. At regular time intervals, the classifier runs a background service that indexes the published documents based on certain properties. These index properties could be simple constraints such as codec types (e.g. H264, G711), geographical location (e.g. America, Asia, Europe), and substrate features (e.g. signaling, mixing). For instance, a Level 1 broker running the classifier can infer the type of substrates based on the substrate features it exposes and it can create an index based on the *atomic substrate types*. A level 2 broker running the classifier can infer type of substrates based on the composite substrates' features they expose and create an index based on *composite substrate types*. An example of the classifier function and index creation in level 1 and level 2 brokers is illustrated in Figure 16. For the inferring facts, an OWL-based reasoner can be used within the classifier component.

The reasoner uses the ontology manager (discussed in the next section) to retrieve the relevant ontologies and rules based on the broker level. In this manner, the classifier component will generate new fact(s) and insert them into appropriate indexes. The inferred facts are later utilized during the discovery requests to the broker, they also help to reduce the response time for requests which will be discussed in detail in section 4.2.2.4. The *#id* indicates the named graph URI (context URI) assigned to the description document soon after the validation phase.

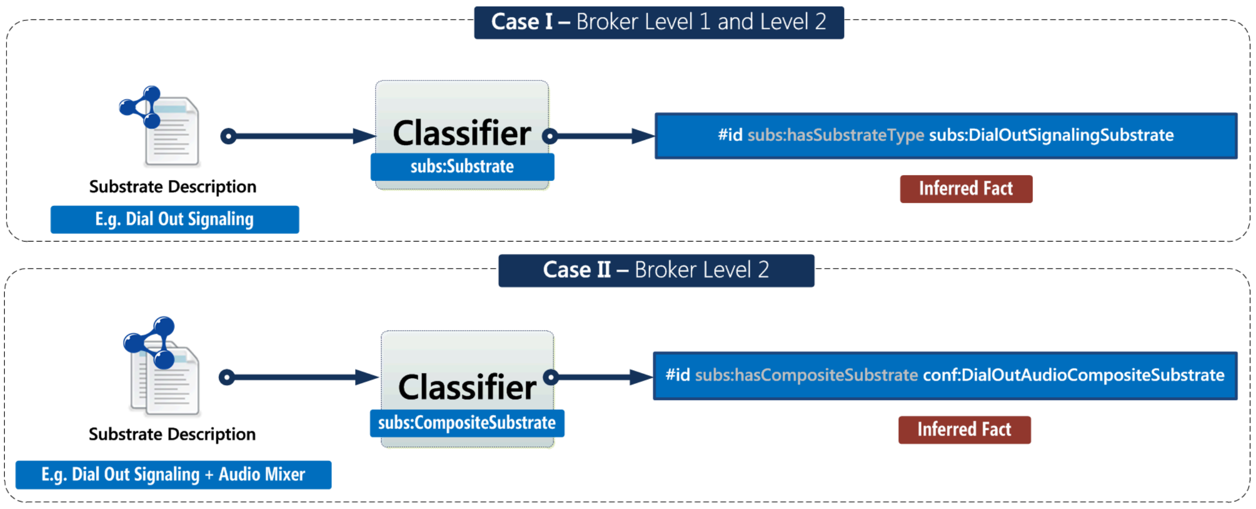

Case I – Broker Level 1 and Level 2
Substrate Description
E.g. Dial Out Signaling
Classifier
subs:Substrate
#id subs:hasSubstrateType subs:DialOutSignalingSubstrate
Inferred Fact
Case II – Broker Level 2
Substrate Description
E.g. Dial Out Signaling + Audio Mixer
Classifier
subs:CompositeSubstrate
#id subs:hasCompositeSubstrate conf:DialOutAudioCompositeSubstrate
Inferred Fact

Figure 16: Classification for Index based on Broker Level and Substrate Type

We chose to index periodically instead of after each publication to optimize the broker's resource usage and uptime. For instance, the indexing may be scheduled for periods where traffic is low, allowing the broker's full capacity to answer the users' requests during busier periods. Algorithm 2 listed below illustrates how the classifier background service is implemented.

```
Algorithm 2 ClassifierComponent_BackgroundService()
1: Generate a unique list of constraints for all the requester profiles $P^c$
2: for each document $d$ of type description in the repository
3:         Get the granularity of the description, $G^d$ = CheckGranularity(d)
4:         Execute InferFactsAndIndex(d , $G^d$) to infer facts and add inferred facts to matching indexes.
```

Algorithm 3 illustrates how the inference rules are loaded based on the description document's type and granularity. It calculates the granularity of the description and executes the inference operation. Granularity indicates the number of substrates present in the substrate document. For example, a composite substrate containing two features (e.g. dial-out signaling and mixing) has a granularity (G^d) of two. The loaded inference rules are later used to generate new fact(s), which are added to the index file.

```
Algorithm 3 InferFactsAndIndex(Description d, Granularity $G^d$)
1: Get the type $T^d$ of the description document.
2: Execute LoadInferenceRulesFromOntologyManager($T^d$, $G^d$) to load the rule set $R$ based on the description
3: for each concept $r_i$ in $R$
4:         Get Inferred triples for the description document $I^d$ = AddInferredFacts(d, $r_i$)
5:         Add $I^d$ to the index file for $r_i$
```

4.2.2.3 Management of the Cloud-based Conferencing Ontology

Ontologies are managed using the ontology manager and ontology crawler. These two components are explained in detail in the following sub-sections.

4.2.2.3.1 Ontology Manager

Ontology manager is used to validate the concepts described in the substrate description document. The ontology manager provides an intuitive interface for adding, removing, updating and retrieving the cloud-based conferencing ontology and other relevant ontologies from the semantic data store. These ontologies are relevant for the efficient validation of description documents by substrate document validator. Besides validation, the ontology manager is also important for the proper inference of new triples by the classifier component. For this reason, the ontology manager maintains a set of inference rules (e.g. a dial-out signaling substrate and an audio mixer substrate make a dial-out audio conference application for a broker level 3), which can be used by the classifier while generating indexes.

4.2.2.3.2 Ontology Crawler

Ontology Crawler is a component responsible for crawling pages to download new ontologies based on the ontology references in the description document. When a description document is published, the ontology references are checked against the ontology manager for validation. In certain cases the description document could reference new upper ontologies that may not already exist in the semantic data store. However, they should also be considered in order to efficiently validate the description document (i.e. checking against native RDF and OWL data types [70] such as literals or integers). The ontology crawler downloads these additional ontologies in order to consider them during validation. Once downloaded, the ontology crawler uses the ontology manager to add these new ontologies into the semantic data store.

4.2.2.4 *Efficient Discovery of Substrates*

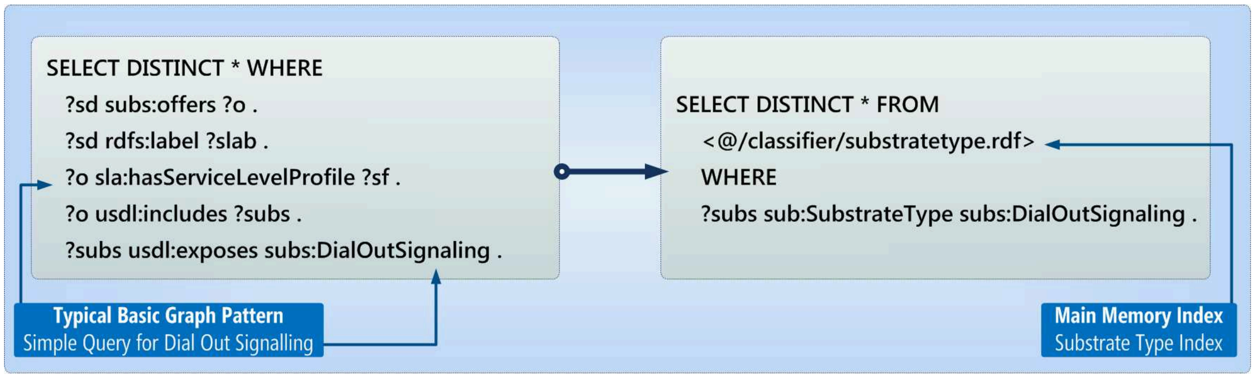
The Query and Ranking engine is responsible for efficient discovery of the substrates. These two components are explained in detail in the following sub-sections.

4.2.2.4.1 Query Engine

The infrastructure provider, platform provider, or conference end-user can look for a substrate by providing the criteria required as content of the REST request. The criteria are specified using the SPARQL specification. Upon receiving the request, the broker uses the query engine to parse the SPARQL query and ensures the request is coherent with the described ontologies. Typical SPARQL consists of a set of basic graph patterns (BGPs) expressed as RDF triples. These RDF triples are then matched against the semantic data store.

Most existing semantic data stores support two modes of storage, query and manipulation of semantic-oriented data: main memory (RAM) and on-disk (e.g. RDBMS). Evidently, the main memory mode for storage, query and manipulation is much faster. We use a main memory mode for treating queries based on the properties that were indexed by the classifier component. The indexes are magnitudes smaller when comparing to an entire semantic data store of description documents. The comparison can be illustrated by considering a sample semantic data store of 200 substrates and 100 substrate providers. In such a situation, the semantic data store will contain approximately 42,000 triples, whereas an index based on particular conference substrate property such as substrate type will contain only 200 triples. Consequently, the un-optimized query will choose to search over a semantic data store of 42,000 triples, the optimized query will search a specific named graph of only 200 triples. In this manner, the query engine is used to optimize the query using SPARQL rewriting rules for BGPs based on these main memory indexes [71] [72]. The cost of execution (by selectivity) is reduced by querying the main memory indexes rather than the entire data store.

The Figure 17 illustrates how a basic query for a substrate type is translated into an optimized query based on the indexes generated by the classifier. Besides such query re-writing techniques, the indexes (based on properties such as substrate type) are stored in main memory to deliver further performance increase. Storing these indexes in the main memory is not a concern due the relative size of each of the indexes in comparison to the entire repository. Currently, our proposed query optimizations only cater to discovery requests (containing basic queries as listed in Figure 17) and do not cater to discovery requests that require ranking.


```
SELECT DISTINCT * WHERE
  ?sd subs:offers ?o .
  ?sd rdfs:label ?slab .
  ?o sla:hasServiceLevelProfile ?sf .
  ?o usdl:includes ?subs .
  ?subs usdl:exposes subs:DialOutSignaling .
```

```
SELECT DISTINCT * FROM
  <@/classifier/substratetype.rdf>
  WHERE
  ?subs sub:SubstrateType subs:DialOutSignaling .
```
Typical Basic Graph Pattern
Simple Query for Dial Out Signalling
Main Memory Index
Substrate Type Index

Figure 17: Query Optimization

4.2.2.4.2 Ranking Engine

The infrastructure provider, platform provider, or conference end-user may wish to limit the number of substrates in the response. For example, an infrastructure provider would want to find the best dial-out signaling substrate giving a set of constraints. In this case the ranking engine uses a ranking algorithm to prioritize the results. As discussed, the ranking engine is not used in conjunction with the query rewriting technique and is mutually exclusive. Weighted Sum Method (WSM) is a Multi Criteria Decision Making (MCDM) technique that is commonly used for ranking the results based on the multiple constraints (e.g. conference participant capacity, delay, bitrate profile, pricing). Of the other MCDM algorithms discussed in section 2.4.4, WSM offered the best alternative with respect to the proposed description mechanism for specifying constraints (e.g. pricing, capacity, availability) pertaining to the conference substrates. For the sake of the simplicity when explaining the function of the ranking algorithm, we are mainly considering numeric constraints.

Consider that the query engine generates an intermediate result set of description documents d_i where $i \in$ [0..n] and n is the number of description documents present in the intermediate result set. WSM Ranking algorithms [73] are applied in two steps viz. 1) Scaling and 2) Calculation of the rank value,

- **Scaling** – The first step of the ranking algorithm is to normalize the values for the constraints using a scaling mechanism. The scaling is required to create a set of normalized values for a dynamic range of values. This is an important pre-requisite for prioritization, since the range of numerical values for the constraints (e.g. latency, response time) is not known beforehand. The scaling should also consider that constraints may show either positive or negative tendencies. Some values such capacity, number of codecs supported, bitrates, and availability indicate better quality for higher values hence displaying a positive tendency. Whereas, values such as price, latency, and response time indicate better quality for lower values, hence displaying a negative tendency.

Consider that the description documents consist of a set of constraints v_j where $j \in$ [0..m] and m is total number of constraints present in a description document *d*. Scaling is implemented typically using the formula for the value of the constraint v_i and a description document *d,*

For positive tendency,

$$Scaled\,value\ v_s = \begin{cases} \dfrac{v_j - v_{min}}{v_{max} - v_{min}} & if\ v_{max} - v_{min} \neq 0 \\ \\ 1 & if\ v_{max} - v_{min} = 0 \end{cases} \tag{1}$$

For negative tendency,

$$Scaled\,value\ v_s = \begin{cases} \dfrac{v_{max} - v_j}{v_{max} - v_{min}} & if\ v_{max} - v_{min} \neq 0 \\ \\ 1 & if\ v_{max} - v_{min} = 0 \end{cases} \tag{2}$$

v_{max} Indicates the maximum value for a particular constraint

v_{min} Indicates the minimum value for a particular constraint

- **Calculation of the rank value –** The second step of the ranking algorithm is to calculate the rank value based on the weightages or nominal values assigned to the constraints by the requester. Each requester is assumed to have assigned the constraint's weightages beforehand**,** using a basic questionnaire.

An example vector of weightages assigned can be as follows,

$$\{price \ = \ 0.6, \qquad latency = \ 0.2, \qquad availability \ = \ 0.1\}$$

According to this vector and the weights are given to the constraints. Sum of the weightages is always one. The order of the constraints based on their importance from the perspective of requester is: first the price (by a 0.6 share from total), then latency (by a 0.2 share from total) and then the availability (by a 0.1 share from total).

We present the mathematical formula for the calculation of the rank value for the description document representing substrates. The weightage is crucial to the calculation of the rank value and is given by w_k where k ∈ [0..r] and r is number of constraints relevant to the requester of discovery. The sum of all the weightages is given by the formula (3) below,

$$\sum_{k=0}^{r} w_k \ \text{ and } \qquad w_k \in [0..1] \tag{3}$$

The following formula (4) calculates the rank value (also referred to as a score) for a description document d_i given the constraint's weightages and scaled values for this document. The constraints' weightages are given by w_k and the scaled values are given by v_{ik}, where *i* is the ordinal of description document in the intermediate result set and k is the ordinal of constraints relevant to the requester.

$$Rank\ Value\ \beta_i = \sum_{k=0}^{r} (v_{ik} * w_k) \tag{4}$$

Once the ranks have been calculated according to formula (4) for each of the description documents, we sort description documents in decreasing order of the rank values. This generates an ordered set of description documents according to the requester's importance.

4.3 End-To-End Publication and Discovery Steps and Specific Illustrative Scenario

In this section, first we will present end-to-end publication and discovery steps to illustrate all the interactions between the different providers and finally the conference end-users. Later, we present a specific illustrative scenario to provide specifics of a single publication and discovery operation.

4.3.1 End-To-End Publication and Discovery Steps

There are a set of seven steps depicted in the Figure 18, for illustrating the end-to-end publication and discovery operations. Steps 1, 3, and 6 denote publication by the substrate providers, infrastructure providers, and service providers, respectively. Steps 2, 5, and 7 denote discovery by the infrastructure provider, platform provider, and end-users, respectively. Step 4 denotes the usage of platform provider's tools by service provider to create a conferencing service. After the creation of the conferencing service, the conference service description is published to level 3 broker by the service provider in the step 6. During publication at each broker level, the providers will alter information in the description before publishing the description to the higher level broker. This publication flow provides flexibility for the providers to add or manipulate functionality or constraints. For example, after discovery from the level 1 broker, the infrastructure provider will alter the offered information such as infrastructure provider information, cost model, etc. before publication to the level 2 broker. Additionally the infrastructure providers may combine multiple substrates before publishing them to the higher-level broker. For example, the infrastructure provider may combine multiple substrates to create a composite substrate before publication to the level 2 broker.

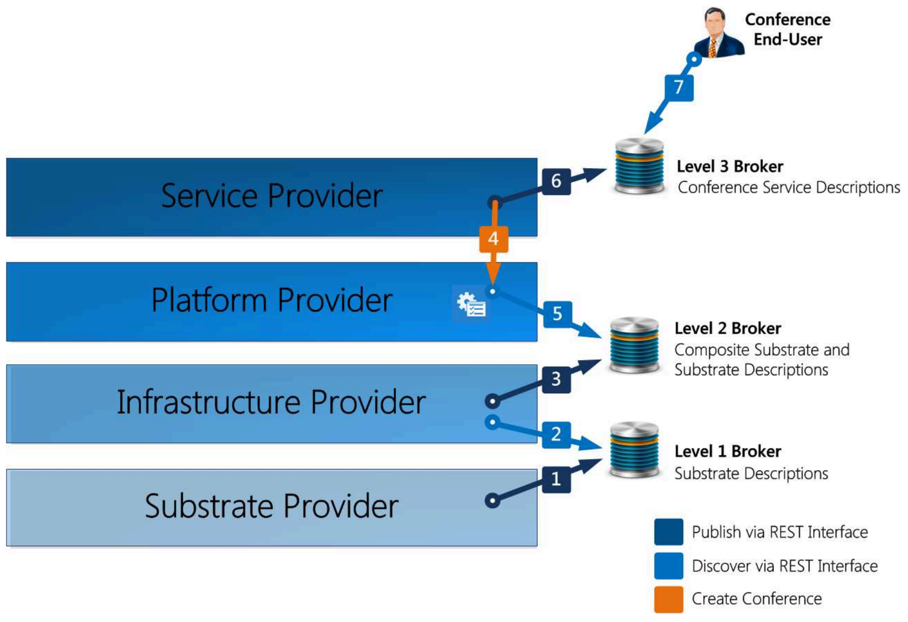


Figure 18: End-To-End Publication and Discovery Steps

4.3.2 Illustrative Scenario

In this section, we will present a specific illustrative scenario for publication and discovery to/from level 1 broker (Steps 1 and 2 in the Figure 18). We will discuss this scenario in detail with respect to the broker components.

The conference substrate providers CSP1 and CSP2, publish the dial-out signaling and audio mixer substrate descriptions to the level 1 broker using RDF XML and RDF N3 serialization formats respectively. The publication is done by sending an HTTP POST request to the broker, where the substrate description is sent as the content of the request. These substrates are described using the proposed semantic-oriented description framework. The substrate providers use the concepts from Level 1 and Level 2 ontology along with the common ontology in order to represent the dial-out signaling substrate and audio mixer substrate. The representation language used to describe these substrates is OWL. Consider that the substrates have the following constraints:

- Dial-out signaling has a price of 9.9 Canadian dollars/month, maximum capacity of 150 participants and latency of 1200 ms
- Audio mixer has a price 19.9 Canadian dollars/month, maximum capacity of 175 participants, latency of 1400 ms, bitrate profile of 128 kbps, and supports codecs-G711 and Speex.

Figure 19 depicts the publication of substrates to the level 1 broker. The dial-out signaling substrate description is directly forwarded to the substrate document validator, because it is already in RDF/XML format, whereas the audio mixer description is first transformed to RDF XML format by the transformation engine. The substrate document validator uses the ontology manager to ensure that the descriptions are valid and conforms to the latest cloud-based conferencing ontology before storing descriptions into the semantic data store.

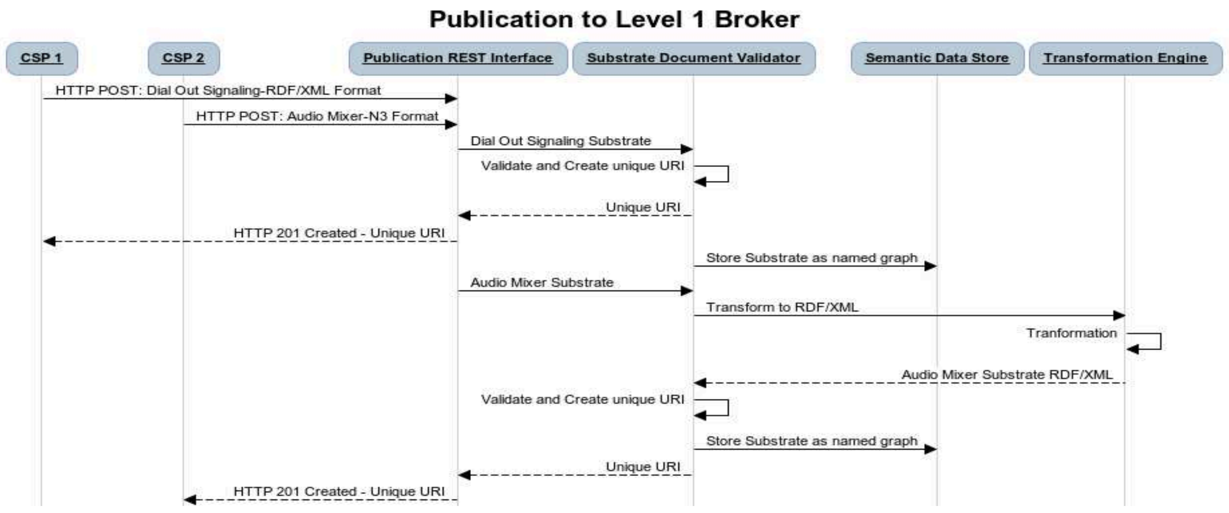

Publication to Level 1 Broker
CSP 1
CSP 2
Publication REST Interface
Substrate Document Validator
Semantic Data Store
Transformation Engine
HTTP POST: Dial Out Signaling-RDF/XML Format
HTTP POST: Audio Mixer-N3 Format
Dial Out Signaling Substrate
Validate and Create unique URI
Unique URI
HTTP 201 Created - Unique URI
Store Substrate as named graph
Audio Mixer Substrate
Transform to RDF/XML
Tranformation
Audio Mixer Substrate RDF/XML
Validate and Create unique URI
Store Substrate as named graph
Unique URI
HTTP 201 Created - Unique URI

Figure 19: Sequence for Publication to Level 1 Broker

Later, the conference infrastructure provider issues two requests to discover a dial-out signaling substrate and an audio mixer substrate in RDF N3 format (illustrated by Figure 20). Let us assume for the sake of the scenario that the infrastructure provider requires ranked discovery results. The discovery requests are expressed in SPARQL and sent as the content of an HTTP GET request. If we assume that the infrastructure provider is more concerned with the price of the substrate and the substrate providers mentioned earlier provide the best pricing model. A sample vector of weightages for the requesting infrastructure provider is assumed to be {price=0.7, latency=0.2, substrate capacity=0.1}. So, according this vector the infrastructure provider is primarily interested in the price of the substrate. Since the query requires ranking, the query engine forwards the requests to the semantic data store and then to the ranking engine. The semantic data store returns the relevant results based on criteria of the SPARQL query. Later, the query engine forwards the request with intermediate set of results to the ranking engine.

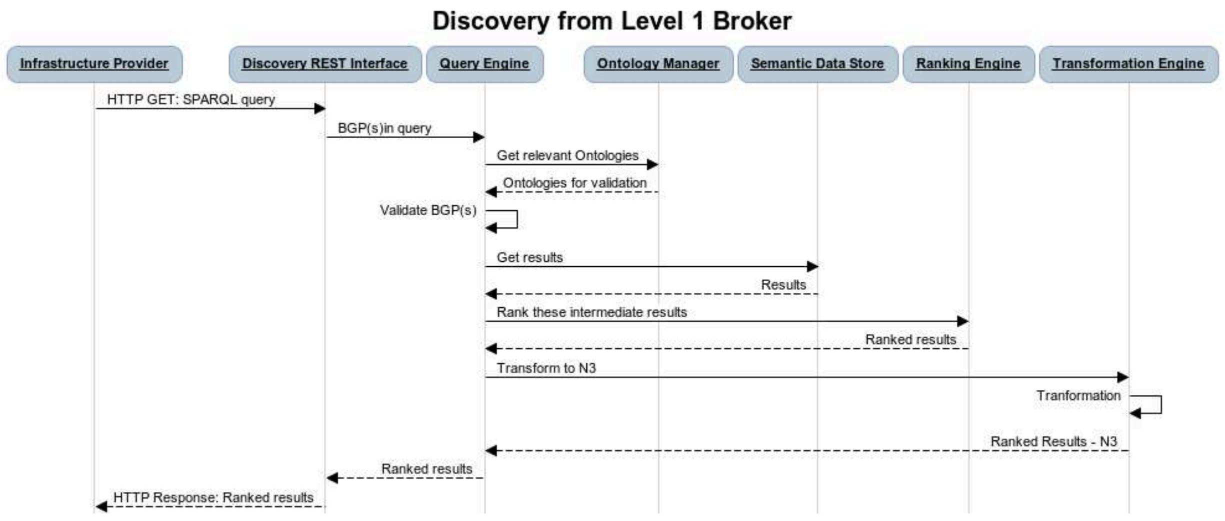


Figure 20: Sequence for Discovery of a Substrate from Level 1 Broker

Soon after, the ranking engine orders the results according to the formula (4) and chooses the top result for each substrate type (i.e. separate results for the dial-out signaling substrate and audio mixer substrate). Finally, the transformation engine transforms the XML substrate descriptions into RDF N3 format and the result is sent to infrastructure provider.

4.4 Chapter Summary

In this chapter we presented the overall architecture with roles and provided an overview of publication and discovery steps by various providers and consumers. We described the proposed semantic-oriented description framework along with the constituent cloud-based conferencing ontology that is part of the framework. The sub-ontologies of the cloud-based conferencing ontology were also explained in detail. Broker architecture was proposed, its components and their functions were explained in detail. These components, not only allow the efficient discovery and publication of conference substrates using standard technologies, but they also provide an extensive architecture supporting multiple description document serialization formats.

We also discussed the end-to-end steps for the publication and discovery operations in cloud-based conferencing. Later, we explained a specific illustrative scenario in detail.

Chapter 5

5 Validation

This chapter validates the proposed architecture; it starts by explaining the overall prototype architecture for publication and discovery of substrates in cloud-based conferencing. Next, we discuss the performance evaluation on the prototype using a benchmarking tool. Benchmarking tool is used for testing and measuring the effectiveness of prototype architecture. Finally, we summarize this chapter.

5.1 Overall Prototype Architecture for Publication and Discovery of Substrates in Cloud-based Conferencing

In this section, we will start by presenting the prototype architecture. Next, we will discuss the software tools used to implement the prototype. Later, we discuss the implemented scenario.

5.1.1 Prototype Architecture

The prototype architecture is depicted in Figure 21 is on the basis of the proposed semantic-oriented description framework and the broker architecture for the publication and discovery of substrates in cloud-based conferencing. As part of the semantic-oriented description framework, we describe conference substrate descriptions using the cloud-based conferencing ontology. As part of the broker, we implemented all the components (Section 4.3), except the *Semantic Ontology Crawler*.

As depicted in the Figure 21, the requesters interact with the broker using these REST APIs. The REST APIs for publication and discovery interfaces are implemented using Jersey JAX-RS server. The broker uses a Sesame-based [74] semantic data store in order to store the conference substrate descriptions, cloud conferencing ontology, indexes, and rules. *Classifier* component uses the Jena reasoner and rules from the Sesame-based semantic data store for adding inference capabilities during the creation of indexes. Sesame-based semantic data store is accessed by the other broker components using REST APIs. These REST APIs are implemented using Jersey JAX-RS client.

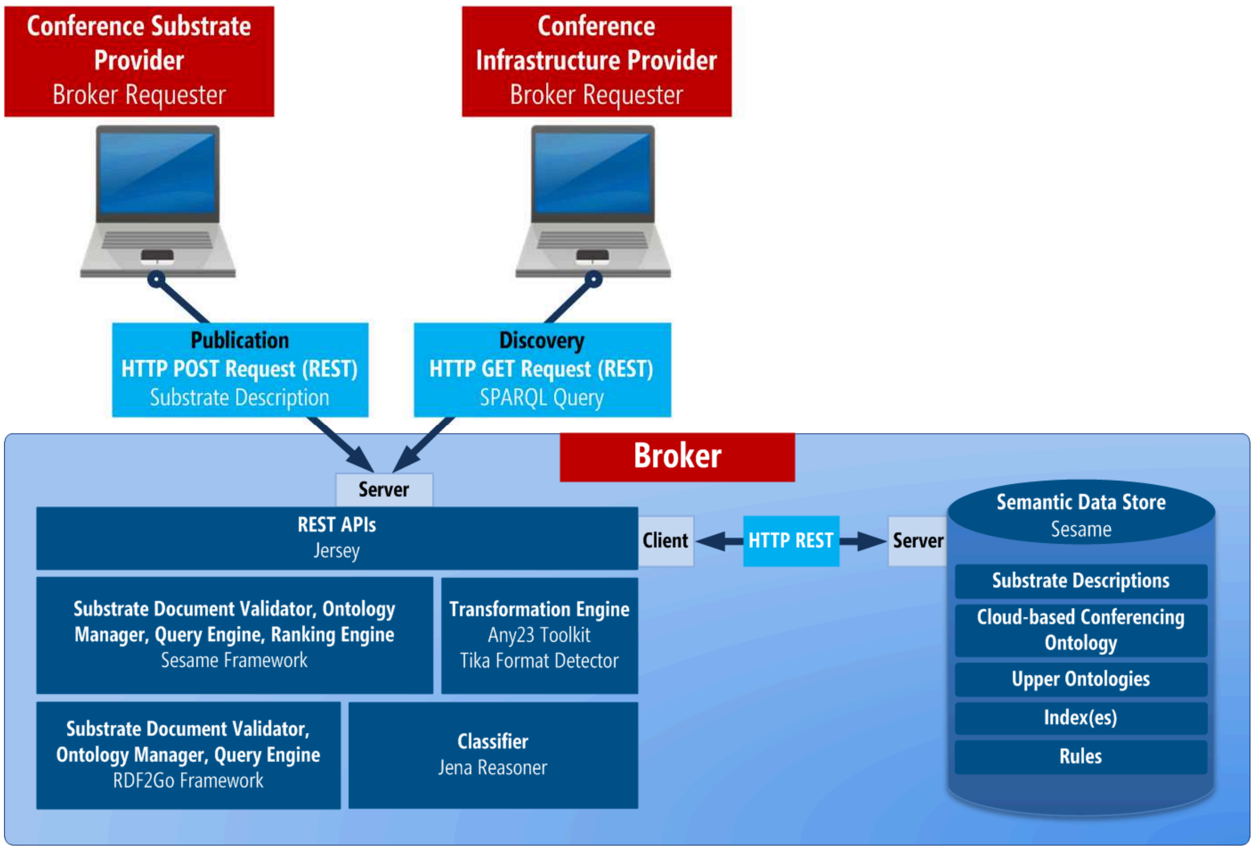


Figure 21: Prototype Architecture

Substrate Document Validator, Ontology Manager, Query Engine, and Ranking Engine components for the management of substrate descriptions, ontologies, querying and ranking operations are implemented using the Sesame framework. The *Transformation Engine* primarily uses two tools viz. Any23 Toolkit and Tika Format Detector. *Substrate Document Validator, Ontology Manager, and Query Engine* additionally use RDF2Go Framework for easier access of the semantic data store. Java is used as the primary programming language to implement this prototype. Eclipse is used as the Integrated Development Environment (IDE) for the development of the prototype.

5.1.2 Software Tools

In this sub-section, we discuss the different software tools used to implement the prototype viz. Jersey [75], Sesame Framework [74], Jena Reasoner [76], Any23 Toolkit [77], RDF2Go Framework [78] and Tika Format Detector [79].

5.1.2.1 Jersey

Jersey [75] provides an open source framework for RESTful web services. It provides support for Java APIs for RESTful Services (JAX-RS). Jersey libraries contain implementation for supporting both RESTful web service client and server. From the perspective of the prototype, we use Jersey server implementation for the publication and discovery interfaces of the broker depicted in the proposed architecture. JAX-RS is a standard specification (JSR 311) for RESTful web services from the Java Community. Besides, we also use Jersey client for accessing the Sesame semantic data store.

5.1.2.2 Sesame Framework

Sesame Framework [74] offers a collection of APIs for parsing and validation, storage, and querying of RDF data based on Java programming language. We use these APIs for the implementation of *Substrate Document Validator, Ontology Manager, Query Engine, and Ranking Engine* components from the proposed architecture.

Substrate Document Validator uses the Sesame RDF I/O (RIO) for parsing and validation of the substrate descriptions. For the prototype, a custom RDF parser is created to validate against cloud-based conference ontology. It allows parsing and validation of the RDF file. Repository API is used to store RDF data in the semantic data store by *Ontology Manager* component.

Sesame framework provides querying support using the standard-based query languages such W3C SPARQL. For the prototype, the *Query Engine* uses Sesame custom query optimization API to re-write and optimize the simple queries. Besides, we developed a custom SPARQL function for ranking of the substrates for the *Ranking Engine*. The parameters to the SPARQL function include the values to be considered while ranking a substrate (e.g. price, latency, availability).

5.1.2.3 Jena Reasoner

Jena Reasoner [76] is a reasoning engine used to derive additional facts from OWL or RDFS description based on a set of rules. We use Jena Reasoner for implementing the *Classifier* component of the proposed architecture. Jena reasoner API provides support for transitive dependencies and user defined rules. Transitive dependencies are required for the cloud-based conferencing ontology. Consider a relationship that *Signaling* is a type of *Substrate* and *DialOutSignaling* is a type of *Signaling.* So, *DialOutSignaling* is also a type of *Substrate.*

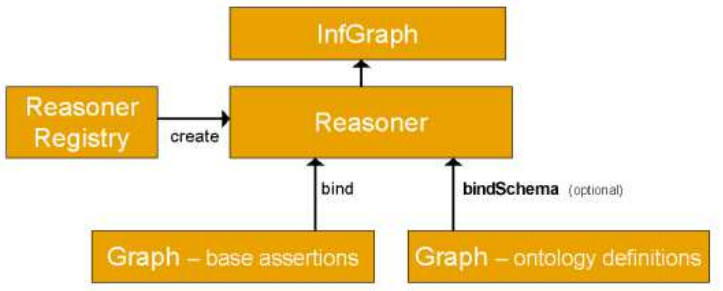
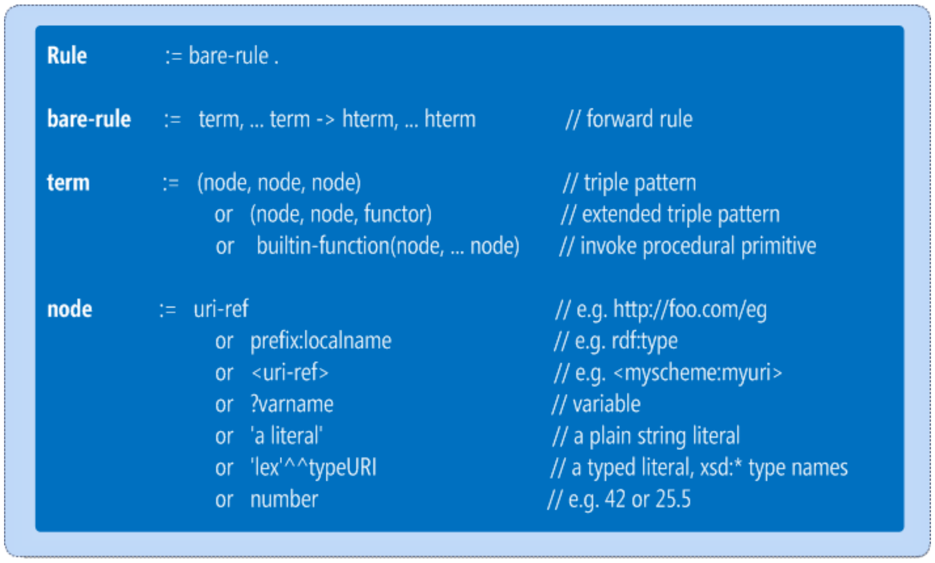


Figure 22: Jena Reasoner API [76]

The Jena reasoner's API (Figure 22) also allows us to derive additional facts based on the cloud-based conferencing ontology, substrate description, and a set of user custom rules. It allows us to bind cloud-based conferencing ontologies using bindSchema function and it allows us to bind the instance data i.e. substrate descriptions via the bind function. User defined rules in Jena inference sub-system, allow us to define rules in two modes of execution viz. forward chaining and backward chaining. We use the forward chaining mode in our prototype. The following abstract syntax [76] (illustrated in Figure 23) is used to define concrete rules in order to inference the conference specifics. In Table 3 below, we list some of the concrete rules defined for the cloud-based conferencing. Such rules are extensively used by the *Classifier* component.

Broker Level	Type	Rule	Associated Axiomatic Triples
Broker Level 1	Dial-Out Signaling Substrate	[dialOutSignalingSubstrateRule: (?SD rdf:type subs:SubstrateDescription) (?SD usdl:offers ?OF) (?OF usdl:includes ?S) (?S rdf:type subs:Substrate) (?S usdl:exposes ?SF) ?SF rdf:type subs:DialOutSignaling -> (?SD subs:hasSubstrateType subs:DialOutSignaling)]	subs:hasSubstrateType rdf:type owl: ObjectProperty . subs:hasSubstrateType rdfs:domain subs:SubstrateDescription . subs:hasSubstrateType rdfs:range subs:Substrate .
Broker Level 2	Dial-Out Audio Mixer Substrate	[dialOutAudioSubstrateRule: (?SD rdf:type subs:PlatformSubstrateDescription) (?SD usdl:offers ?OF) (?OF usdl:includes ?CS) (?CS rdf:type subs:CompositeSubstrate) (?CS usdl:exposes ?SL) listLength(?SL, ?len) equal(?len,2) listContains(?SL, subs:DialOutSignaling) listContains(?SL, subs:AudioMixer) -> (?SD subs: hasCompositeSubstrateType subs:DialOutAudioSubstrate)]	conf:DialOutAudioSubstrate rdfs:subClassOf subs:CompositeSubstrate . subs:hasCompositeSubstrateType rdfs:subPropertyOf subs:hasSubstrateType . subs:hasCompositeSubstrateType rdfs:domain subs:PlatformSubstrateDescription . subs:hasCompositeSubstrateType rdfs:range conf:CompositeSubstrate .
Broker Level 3	Dial-Out Audio Conference with Moderator Support	[dialOutAudioConferenceWithModeratorRule: (?SD rdf:type subs:ServiceDescription) (?SD usdl:offers ?OF) (?OF usdl:includes ?CS) (?CS rdf:type subs:CompositeSubstrate) (?CS usdl:exposes ?SL) listLength(?SL, ?len) equal(?len,3) listContains(?SL, subs:DialOutSignaling) listContains(?SL, subs:AudioMixer) listContains(?SL, subs:FloorControl) -> (?SD conf: hasConferenceType conf:DialOutAudioMixerWithModeratorConference)]	conf:DialOutAudioMixerWithModeratorConference rdfs:subClassOf conf:Conference . conf:hasConferenceType rdfs:subPropertyOf subs:hasCompositeSubstrateType . conf:hasConferenceType rdfs:domain subs:ServiceDescription . conf:hasConferenceType rdfs:range conf:Conference .

Table 3: Sample Custom Rules for Reasoner for Broker Level 1, 2 and 3



```
Rule        := bare-rule .

bare-rule   :=  term, ... term -> hterm, ... hterm          // forward rule

term        :=  (node, node, node)                          // triple pattern
                 or  (node, node, functor)                  // extended triple pattern
                 or   builtin-function(node, ... node)      // invoke procedural primitive

node        :=  uri-ref                                     // e.g. http://foo.com/eg
                 or  prefix:localname                       // e.g. rdf:type
                 or  <uri-ref>                              // e.g. <myscheme:myuri>
                 or  ?varname                               // variable
                 or  'a literal'                            // a plain string literal
                 or  'lex'^^typeURI                         // a typed literal, xsd:* type names
                 or  number                                 // e.g. 42 or 25.5
```

Figure 23: Abstract Syntax for User-defined Rules [76]

5.1.2.4 Any23 Toolkit

Apache Any23 Toolkit [77] is a toolkit which allows conversion between popular semantic-oriented serializations formats such RDF/XML, RDF/N3, RDF/JSON, RDFa, Microformats, and also common formats such as Comma Separated Values (CSV). The toolkit contains a library, REST-based Web Service [77] and a command line tool. For the purpose of the prototype, we are using the Any23 toolkit as part of the *Transformation Engine* component of the proposed architecture to allow conversion of popular formats to RDF/XML. This allows the providers to publish and allows the requesters to discover the description document(s) in the format that is compatible with their existing tools.

5.1.2.5 RDF2Go Framework

RDF2Go [78] provides a Java-based semantic framework for abstraction over triple store such as Sesame [80]. It allows developers to create programs agnostic of the underlying semantic data store. Besides, it also offers easier management of the named RDF graphs and querying within a graph. RDF2Go is extensively used by the *Ontology Manager*, *Substrate Document Validator,* and *Query Engine* components of the proposed architecture.

5.1.2.6 Tika Format Detector

Apache Tika Format Detector [79] is a collection of libraries that enable extraction of metadata and structured textual information from various file types. It is used primarily during publication phase to detect the format of the submitted conference substrate description. By doing so, it aids the *Transformation Engine* to decide the type of transformer to use (e.g. N3 to RDF/XML Transformer, Turtle to RDF/XML Transformer).

5.1.3 Implemented Scenario

In our implemented scenario, the conference substrate providers publish the substrates to the level 1 broker and an infrastructure provider discovers the substrates from the level 1 broker. In addition, the infrastructure provider composes these conference substrates and publishes the composed conference substrate to the level 2 broker. In this section, we discuss our implemented scenarios in three steps. First, we discuss the publication of a dial-out signaling and audio mixer substrate to level 1 broker by conference substrate providers. Second, we discuss the discovery of these conference substrates from level 1 broker by a conference infrastructure provider. Third, we discuss a conference infrastructure provider that composes these conference substrates as a dial-out audio mixer substrate and publishes it to the level 2 broker.

5.1.3.1 Publication of Conference Substrates to Level 1 Broker by Conference Substrate Providers

Conference substrate providers CSP1 and CSP2 publish dial-out signaling substrate (S1) and audio mixer substrate (S2) to the level 1 broker. We send these publication requests using REST. These substrates are described using the proposed semantic-oriented description framework and send as the content of HTTP POST requests. In the Table 4, we give a segment of the audio mixer substrate (S2) description using the RDF/Turtle serialization format. In this segment, we give the CSP2's information, substrate offerings, audio mixer functional feature, substrate interfaces and interaction protocol, constraints, and price plan specification. A more detailed version for audio mixer substrate description can be found in appendix 1.

A Segment of an Audio Mixer Substrate (S2)Description Published by CSP2	

Provider Information	`<http://www.csp2.com/substrates/77554389-935a-4f74-8a59-54fc9d410321/description/>`
	`rdf:type subs:SubstrateDescription ;`
	`   dcterms:title "Audio Mixer Substrate Description";`
	`   dcterms:creator :CSP2 ;`
	`   usdl:hasProvider :CSP2;`
	`:CSP2 a cloud:SubstrateProvider ;`
	`   gr:legalName "CSP2 Communications Inc." ;`
	`   foaf:page <http://www.csp2.com/> ;`
	`   org:address [ a org:PostalAddress ;`
	`            org:streetAddress "1000 Rue Berri" ;`
	`            org:postalCode "H7N 4G9" ;`
	`            org:addressLocality "Montreal, Canada" ] ;`
	`   org:telephone "+1(514)-537-5037" ;`
	`   org:faxNumber "+1(514)-537-5038" ;`
	`   org:email "contact@csp2.com" .`

CSP2 Substrate Offering	:AudioMixerSubstrateOffering a usdl:ServiceOffering;
	usdl:includes :S2;
	usdl:hasPricePlan :CSP2StandardPlan;
	usdl:validFrom "2012-05-19T19:08:24.798Z"^^xsd:dateTime;
	usdl:validThrough "2015-05-19T19:08:24.798Z"^^xsd:dateTime;
	sla:hasServiceLevelProfile :AudioMixerSubstrateProfile .

Audio Mixer Functional Feature	:S2 a subs:Substrate; usdl:exposes subs:AudioMixer ; usdl:hasProvider :CSP2 ; usdl:hasInteractionProtocol :AudioMixerInteractionProtocol ; usdl:hasServiceModel :AudioMixerSubstrateModel .
Interfaces and Interaction Protocol (REST-based Substrate Interfaces)	:AudioMixerInterface a usdl:Interaction ; dcterms:title "REST-based Interactions" ; dcterms:description "End Point details for REST-based Interactions" ; usdl:hasInterfaceOperation :ReserveMix ; usdl:hasInterfaceOperation :GetMixInfo ; usdl:hasInterfaceOperation :RemoveMix ; usdl:hasInterfaceOperation :ReserveParticipantMix ; usdl:hasInterfaceOperation :GetParticipantMixInfo ; usdl:hasInterfaceOperation :StartStopParticipantMedia ; usdl:hasInterfaceOperation :DeleteParticipantMix ; usdl:hasInterfaceOperation :ChangeParticipantMix .
Get Participant's Mix Media Information Interface	:GetParticipantMixInfo a sarest:Operation ; dcterms:title "Get a particular participant's mix info" ; dc:description "Get a particular participant's mix info, specifically current media session information." ; sarest:hasAddress "Mix/{MixId}/Participant/{ParticipantId}"^^sarest:URITemplate ; subs:supportsDataFormats "application/json, application/xml, text/plain"^^rdf:literal ; subs:hasRequestUriParameter <http://purl.org/ontology/conference#MixId> ;

	subs:hasRequestUriParameter <http://purl.org/ontology/conference#ParticipantId> ; subs:affectedResources <http://purl.org/ontology/conference#Mix> ; sarest:hasMethod "GET"^^sarest:HTTPMethod ; subs:hasResponseCode "200 OK"^^http:StatusCode ; subs:hasResponseBody <http://purl.org/ontology/conference#MediaDescriptor> .
Constraints	:MixingCapacity a sla:Variable; rdfs:label "Maximum Mixing Capacity"; sla:hasDefault [a gr:QuantitativeValueInteger ; gr:hasValue “160"^^xsd:integer ; gr:unitOfMeasurement "users"] . :MixerSupportedAudioCodecs a sla:Variable; rdfs:label "Audio Codecs"; sla:hasDefault [a gr:QualitativeValue ; gr:hasValue "G711,Speex"] .

Price Plan	:CSP2StandardPlan a usdl-price:PricePlan;
	dcterms:title "Standard Plan";
	dcterms:description "Standard Plan that fits most business users." ;
	usdl-price:hasPriceCap [a gr:PriceSpecification;
	gr:hasCurrency "CAD";
	gr:hasCurrencyValue "29.99"^^xsd:float;
	gr:unitOfMeasurement "per/month"];
	usdl-price:hasPriceFloor [a gr:PriceSpecification;
	gr:hasCurrency "CAD";
	gr:hasValueFloat "9.99"^^xsd:float;
	gr:unitOfMeasurement "per/month"];
	usdl-price:hasPriceComponent [a usdl-price:PriceComponent;
	dcterms:title "Monthly rate";

	```
	                        dcterms:description "Monthly fee to pay for the substrate.";
	                        gr:hasCurrency "CAD";
	                        gr:hasCurrencyValue "0.25"^^xsd:float;
	                        gr:unitOfMeasurement "per month per user"
	                      ];
	usdl-price:hasTax          [ a usdl-price:PriceComponent;
	                        dcterms:title "Total Tax";
	                        dcterms:description "Provincial and Federal tax component";
	                        gr:hasValueFloat "15.0"^^xsd:float;
	                        gr:unitOfMeasurement "percent"
	                      ] .
	```

Table 4: A Segment from Audio Mixer Substrate (S2) Description Published by CSP2

5.1.3.2 Discovery of Conference Substrates from to Level 1 Broker by Conference Infrastructure Provider

After publication, conference infrastructure provider CIP1 discovers both dial-out signaling substrate and audio mixer substrate (having a maximum capacity of 150 users) using two discovery requests. We restrict the discovery result to the earlier published substrates (S1 and S2) by giving substrate providers' name (CSP1 and CSP2), substrates' functional features, constraint (i.e. maximum capacity) in the discovery requests. We send these discovery requests using REST. The discovery queries for a dial-out signaling substrate (Q1) and an audio mixer substrate (Q2) are expressed using SPARQL, and sent as the content of HTTP GET requests. Upon sending discovery requests, conference infrastructure provider receives the descriptions representing dial-out signaling (S1) and audio mixer substrates (S2), respectively from the level 1 broker.

In the Table 5, we give the SPARQL query Q2 for discovering an audio mixer substrate with maximum capacity greater than 150 users which is provided by CSP2. In this query, we give the namespaces and the selection clause. SPARQL query consists of set of triple patterns called basic graph pattern. SPARQL triple patterns are 3-tuples that could contain variables in the place of subject, predicate, or object. SPARQL variables are indicated by a preceding question mark symbol (?). Every complete triple pattern contains a dot (.) at the end. Triple patterns reusing the subject are specified by a semi-colon (;).

SPARQL Query Q2 for discovery of an Audio Mixer Substrate with maximum capacity 150 which is provided by CSP2	
Namespaces	PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
	PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
	PREFIX usdl:<http://www.linked-usdl.org/ns/usdl-core#>
	PREFIX subs:<http://purl.org/ontology/substrate-description#>
	PREFIX usdl-sla:<http://www.linked-usdl.org/ns/usdl-servicelevel#>
	PREFIX usdl-price:<http://www.linked-usdl.org/ns/usdl-pricing#>
	PREFIX dc:<http://purl.org/dc/elements/1.1/>
	PREFIX gr:<http://purl.org/goodrelations/v1#>
SELECT clause	SELECT DISTINCT ?source WHERE {
	?sd rdf:type subs:SubstrateDescription ;
	dc:source ?source ;
	usdl:offers ?o .
	?o rdf:type usdl:ServiceOffering ;
	usdl-sla:hasServiceLevelProfile ?sf ;
	usdl:hasPricePlan> ?pp ;
	usdl:includes ?subs .
	?sf rdf:type usdl-sla:ServiceLevelProfile ;
	usdl-sla:hasServiceLevel ?sl ;

	usdl:hasProvider :CSP2 . ?subs usdl:exposes subs:AudioMixer . ?sl dc:title ?slvtitle . FILTER regex(?slvtitle, "Capacity") . ?sl rdf:type usdl-sla:GuaranteedState ; usdl-sla:serviceLevelExpression ?sle . ?sle rdf:type usdl-sla:ServiceLevelExpression ; usdl-sla:hasVariable ?slv . ?slv usdl-sla:hasDefault ?valueexpr . ?valueexpr gr:hasValue ?capacityvalue . FILTER (xsd:integer(?capacityvalue) > 150) .

Table 5: SPARQL Query to Discover an Audio Mixer Substrate

5.1.3.3 Publication of Conference Substrate Description to Level 2 Broker

After conference Infrastructure Provider CIP1 discovers the dial-out signaling (S1) and audio mixer (S2) substrates from level 1 broker, it composes the substrate descriptions and creates a dial-out audio mixer substrate (CS1) before publishing it to the level 2 broker. In Table 6, we capture a segment of the CS1 description using the RDF/Turtle serialization format. In this segment, we express the dial-out audio mixer substrate's functional feature collection (*:DialOutAudioMixerFeatures*) as a list of functional features from dial-out signaling substrate (S1) and audio mixer substrate (S2).

A Segment of a Dial-Out Audio Mixer Substrate (CS1) Description Published by CIP1	

Provider Information	<http://cip1.com/substrates/b077dc4b-fa46-4c15-8170-ff85f9a0649d/description/> rdf:type subs:PlatformSubstrateDescription ;
	dcterms:title "Dial-Out Audio Mixer Substrate Description";
	dcterms:creator :CIP1;

	usdl:hasProvider :CIP1; :CIP1a cloud:SubstrateProvider ; gr:legalName "CIP1 Inc." ; foaf:page <http://www.CIP1.com/> ; org:address [a org:PostalAddress ; org:streetAddress "1070 Noble Street" ; org:postalCode "J2D-8P7" ; org:addressLocality "Quebec, Canada"] ; org:telephone "+1(514)-342-2543" ; org:faxNumber "+1(514)-342-2343" ; org:email "contact@cip1.com" .
Dial-Out Audio Mixer Functional Feature Collection	:CS1 a subs:CompositeSubstrate; usdl:exposes :DialOutAudioFeatures ; usdl:hasProvider :CIP1 ; usdl:hasInteractionProtocol :DialOutAudioInteractionProtocol ; usdl:hasServiceModel : DialOutAudioSubstrateModel . :DialOutAudioMixerFeatures a subs:SubstrateFeatureCollection ; rdf:first subs:DialOutSignaling ; rdf:rest :RestOfListDialOutAudioMixerFeature . : RestOfListDialOutAudioMixerFeature rdf:first subs:AudioMixer; rdf:rest rdf:nil .

Table 6: A Segment from a Dial-Out Audio Mixer Substrate (CS1) Description Published by CIP1

5.2 Performance Evaluation

As part of the performance evaluation, two laptops were used to run the prototype. The first one is used to run the benchmarking tool for publication and discovery, while the second one is used to run the broker. The setup of the prototype test-bed is illustrated by the Figure 24. The detailed configuration of the laptops used for prototype test-bed is listed in Table 7.

Laptop 1: Benchmarking tool	
Processor	Intel Core 2 Duo 1.8 Ghz
Operating System	Ubuntu 12.10 x64
Main Memory	4GB DDR2 RAM
Java SDK	Java EE 6
Laptop 2: Broker	
Processor	Intel Core i7 – 2670QM 2.2 Ghz
Operating System	Windows 7 x64
Main Memory	8GB DDR3 RAM
Java SDK	Java EE 7
Web Server & Servlet Container	Tomcat v7

Table 7: Prototype Test-bed Configuration

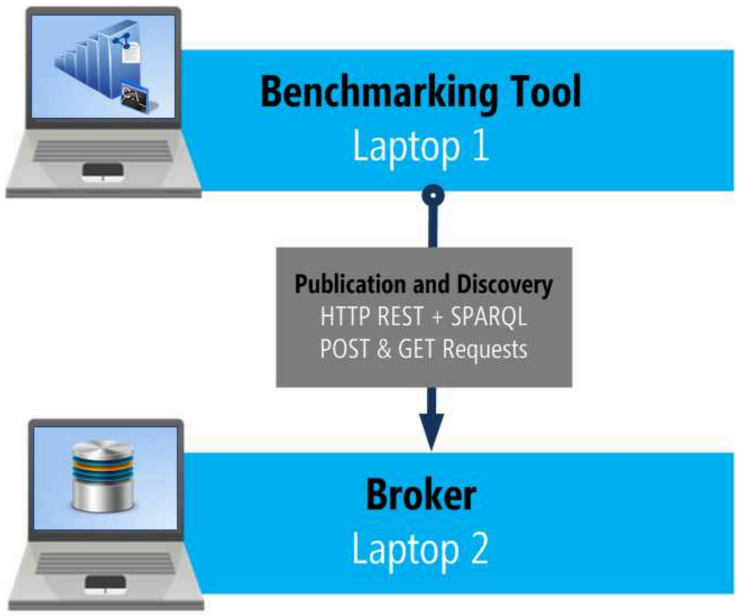


Figure 24: Prototype Test-bed Set-up

In the following sub-sections, we define the benchmarking tool used for testing and measuring the effectiveness of prototype architecture. Later, we discuss the performance metrics and results obtained using the prototype test-bed configuration.

5.2.1 Benchmarking Tool

To have a near realistic view of proposed broker's publication and discovery operations, we needed a test bed set-up with several dozens of substrates belonging to different providers and having various constraints types (e.g. *performance type*: latency and response time, *capacity type*: maximum signaling capacity and mixing capacity, *quality type*: supported codecs and bitrate profile). We also needed several infrastructure providers to issue a mix of queries of varying complexity levels. It is worth mentioning that there are some existing benchmarking tools for RDF-based repositories, but they did not meet the requirements for our prototype. The Berlin SPARQL Benchmark [81] for instance, allows the benchmarking of only pre-defined use cases.

However, we required to generate random traffic for publication and discovery of conference substrates. We therefore implemented a benchmarking tool including a test data generator and query generator. Test data generator simulates conference substrate providers by generating several conference substrate descriptions and publishing them to the broker. While, the query generator simulates a conference infrastructure provider by generating multiple random discovery requests and issues them to the broker. Both generators are implemented as a command line interfaces (CLI) using Java concurrency API and is capable of issuing varying numbers of parallel requests to the broker. The benchmarking tool supports several switches (options) (Table 8).

Switch Name	Switch	Value Type
Broker end-point	endpoint	String (URL)
Number of Users	users	Integer
Number of Parallel Requests	parallelrequests	Integer
Optimize Discovery Query	optimize	Boolean
Run in Batch mode	batchmode	Boolean
No of Batch Runs	runs	Integer
Average-out Batch Run Time	avgtime	Boolean
Discovery Mode	discover	Boolean
Publication Mode	publish	Boolean
Query mix directory	querymix	String (Directory Name)

Table 8: Benchmarking Tool Switches

5.2.2 Performance Metrics

The performance of our prototype is assessed in terms of time delays for both publication and discovery on level 1 broker. The publication (discovery) delay is the time difference between when the substrate (infrastructure) provider sends a publication (discovery) request to the broker and when it receives the response. The delays are measured in milliseconds. The publication delay measurements were taken for different numbers of substrate providers, different number of simultaneous requests, and for the cases where different numbers of substrates were published prior to the time of measurements. The discovery delays were measured for two types of queries: simple and complex queries.

Simple queries are, for instance, those ones based only on the substrate type. An example of simply query is to find the list of available audio mixer substrates. Complex queries may include multiple relational criteria (e.g. capacity>=100 and latency<=1000ms), textual operations (e.g. textual search for a specific provider or substrate within a specific region), or ranking criteria (e.g. get an ordered list of the first 10 recommended audio mixers in Canada). We also compared the discovery delays of simple queries with and without optimization to show the added value of the used optimization algorithms. The optimization algorithms are performed based on the query re-rewriting techniques discussed in section 4.3.2.4.1.

5.2.3 Performance Results

Figure 25, Figure 26, and Figure 27 illustrates the measured results. Corresponding to each of these measured results, Table 9, Table 10, and Table 11 lists the recorded time in milliseconds. Each of these measurements is calculated as an average of 15 experiments. Figure 25 displays the measurements for publishing up to 32 substrates simultaneously by varying the number of existing substrates in the semantic data store. As expected, the delays increase with the number of simultaneous publications as well as the number of substrates already in the registry. Nevertheless, the delays remain acceptable considering that the publication is a one-time operation performed by the substrate providers. The discovery delay measurements were performed on a broker containing 100 substrates. The discovery requests are randomly generated by the benchmarking tool, according to the chosen request complexity (i.e. simple or complex).

Figure 26 compares the discovery delays for optimized and non-optimized simple queries. The results show that the optimization reduces the delays by about 7%, and this percentage can be further increased by creating additional indexes for frequently used basic graph patterns, such as substrate provider region. Complex discovery queries require more processing time and induce much larger delays compared to the simple queries (Figure 27).

Parallel Requests	Delays (milliseconds)				
	No. of existing SDs in repository 0	No. of existing SDs in repository 50	No. of existing SDs in repository 100	No. of existing SDs in repository 150	No. of existing SDs in repository 200
2	860	1880	2822	3809	4667
4	925	2444	4291	6286	8078
8	1419	4825	8699	12471	16299
16	2464	10240	17776	25555	33154
32	7860	23608	38590	53814	69179

Table 9: Publication Delays Summary

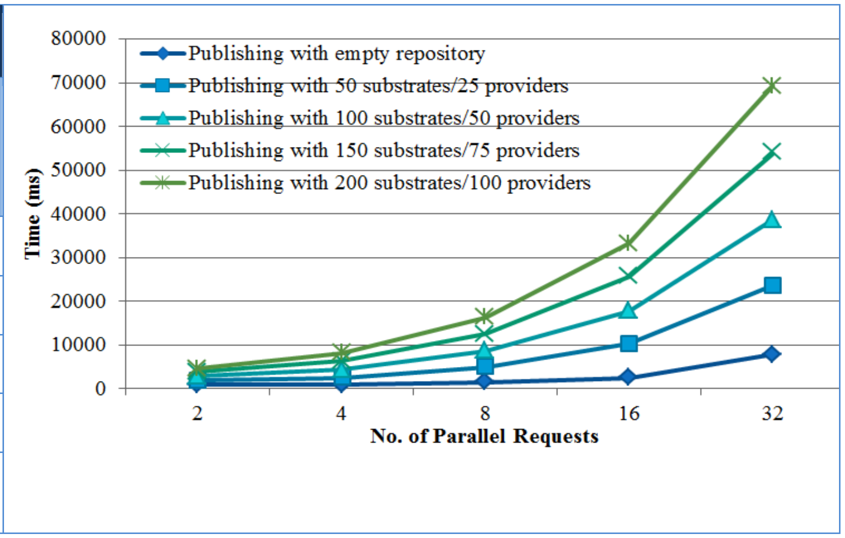


Figure 25: Publication Delays Summary

Parallel Requests	Delays (milliseconds)	
	No. of existing SDs in repository 100	
	Non-Optimized Query	Optimized Query
2	114	106
4	170	160
8	240	218
16	357	335
32	468	450

Table 10: Discovery Delays Summary for Simple Queries

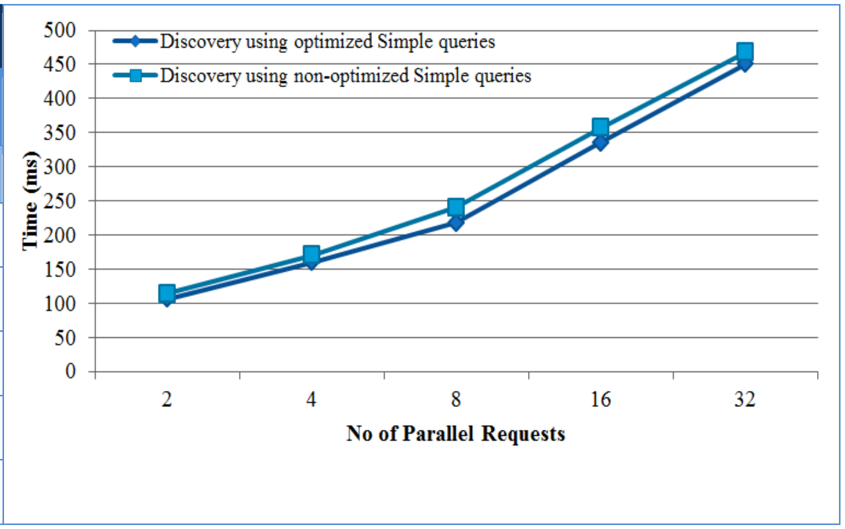


Figure 26: Discovery Delays Summary for Simple Queries

	Delays (milliseconds)
Parallel Requests	No. of existing SDs in repository 100
2	2152
4	3348
8	5347
16	8413
32	16283

Table 11: Discovery Delays Summary for Complex Queries

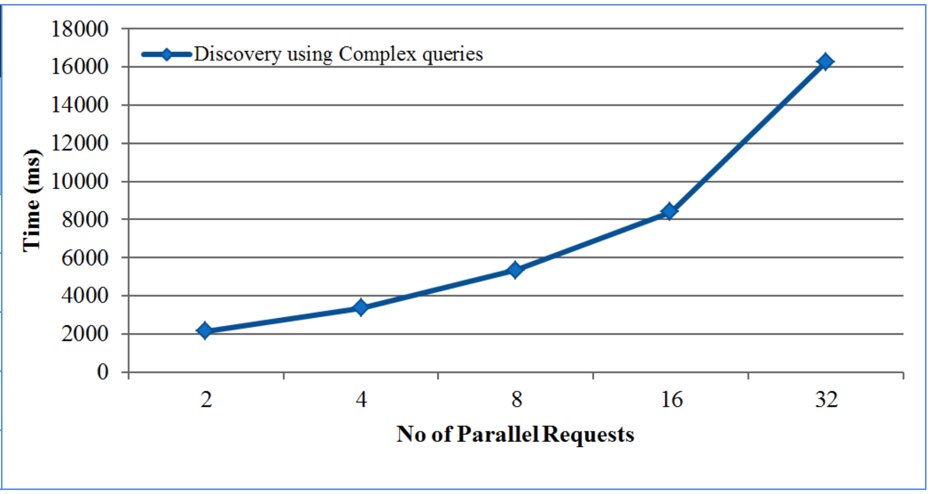


Figure 27: Discovery Delays Summary for Complex Queries

5.3 Chapter Summary

In this chapter we presented the overall prototype architecture for publication and discovery of conference substrates. The prototype architecture contains a broker that enabled the efficient publication and discovery of the cloud-based conference substrates using the constituent software tools. We explained the six constituent software tools in detail. We also discussed regarding the implemented scenario in three steps and provided sample publication and discovery requests. Finally, we presented the performance evaluation using the benchmarking tool with various experimental scenarios. The experimental scenarios varied by having different number of publication and discovery requests, and having various numbers of existing substrates in the semantic data store. Measurements showed that that delay increases with the number of existing substrates in the semantic data store. Also, our measurements on discovery requests showed that the optimized queries display a better performance in comparison with non-optimized queries.

Chapter 6

6 Conclusion and Future Work

This chapter concludes the thesis by providing a summary of contributions and also providing the directions for the future research work.

6.1 Summary of Contributions

Cloud-based conferencing services can provide potential benefits such as the easy introduction of different types of conferences, resource usage efficiency and scalability. A novel business model [3] for cloud-based conferencing was proposed recently. A key role in this model is played by the conference substrate providers. Conference substrates are virtualized and act as elementary building blocks (e.g. dial-out signaling, audio mixing) of conferencing applications. These conference substrates can be shared for resource efficiency purposes. To enable cloud conferencing and usage of conference substrates, substrate providers should describe their conference substrates and make them known by publishing them to a broker so that the consumers/requesters may later discover the conference substrates accurately using specific business and technical aspects.

As part of the contributions to this thesis, we have identified the requirements for a semantic-oriented description framework to describe conference substrates. We have also identified the requirements for a broker architecture in order to enable the publication and discovery of substrates in cloud-based conferencing. Our semantic-oriented description framework consists of two components: cloud conferencing ontology and a description language. Several semantic-oriented description frameworks were evaluated based on our requirements for describing cloud-based conferencing substrates. Besides, we have also evaluated the available service description languages and related ontologies (cloud computing and conferencing ontologies) separately. However, to the best of our knowledge, none of the existing frameworks were capable of meeting all of our requirements. We have also evaluated several prominent broker architectures for publication and discovery based on our requirements and observed that none of them satisfy all of our requirements.

This thesis proposed a semantic-oriented description framework which includes a cloud-based conferencing ontology using OWL as the description language. It enables the specification of both technical and business aspects of conferencing substrates. Furthermore, we designed a broker architecture for publication and discovery of substrates in cloud-based conferencing. The providers and requesters communicate with the broker via the REST API.

A proof of concept prototype was implemented based on the proposed semantic-oriented description framework, and a broker architecture for the publication and discovery of substrates in cloud-based conferencing. To validate our prototype we used a benchmarking tool, which was specifically created for the purpose of testing the performance of publication and discovery of conference substrates by generating random traffic to the broker and measuring the key parameters such as publication and discovery delays. For both publication and discovery, the performance results showed that the delay increases with the number of existing conference substrate descriptions in the semantic data store. The performance results showed satisfactory results for publication even considering the additional validation steps that were performed.

Furthermore, the performance results showed that the query engine optimizations reduced the delays for certain discovery requests (when using simple queries) by about 7%.

6.2 Future Work

We have identified some research items to be considered. We start by discussing the future research directions relating to the proposed semantic-oriented description framework and later discussing the future research directions relating to the broker architecture for publication and discovery of substrates in cloud-based conferencing.

6.2.1 Semantic-oriented Description Framework

For the semantic-oriented description framework, we have identified two key research directions; one relates to integration of substrate descriptions with composition language and the other relates to description of complex substrate features. For enabling the orchestration and choreography of constituent substrates of conferencing application, one of the major challenges is the integration of the existing composition languages (such as jOpera) with the conference substrate descriptions as input. Another challenge relates to identifying the description of complex conference substrate features such as transcoding, virtual network computing (or desktop sharing) based on the proposed semantic-oriented description framework.

6.2.2 Broker Architecture for Publication and Discovery of Substrates in Cloud-based Conferencing

For the broker architecture, we have identified three key research directions; one relates to query optimization, next relates to performance of the semantic data store and last relates to the distributed broker architectures. The proposed broker architecture consists of a query engine component that supports query optimizations for simple queries. Query optimizations for complex queries needs to be further investigated to improve the overall discovery delays. The proposed broker architecture uses semantic data store for the storage of conference substrate descriptions (as triples). Further investigation is required for evaluating the performance of the semantic data store for larger repository sizes (of magnitude greater than 1 million triples) using alternatives such as NoSQL catering specifically to larger datasets. Finally, further research is needed to allow discovery of conference substrates from multiple distributed brokers. Furthermore, individual broker components (such as semantic data store) can be distributed for achieving scalability and efficiency of resources. By exploiting the nature of RDF and SPARQL, we can query multiple diverse and distributed semantic data store components. However, there are challenges such as synchrony of parallel read-write operations and locking of specific named graphs for manipulation.

Appendix

Appendix A: Audio Mixer Substrate (S2) Description Published by CSP2 in Turtle Serialization Format

```
#
# Audio Mixer Substrate (S2) Description Published by Conference Substrate Provider - CSP2
#


#
# Initial Prefix Information
#
@prefix : <http://www.[companynamewebsite]/substrates/[uuid1]/description/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix subs: <http://purl.org/ontology/substrate-description#> .
@prefix cloud: <http://purl.org/ontology/cloud-conference-infrastructure#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sawsdl: <http://www.w3.org/ns/sawsdl#> .
@prefix gr: <http://purl.org/goodrelations/v1#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix org: <http://schema.org/Organization#> .
@prefix ctag: <http://commontag.org/ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix http: <http://www.w3.org/2011/http#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix usdl: <http://www.linked-usdl.org/ns/usdl-core#> .
@prefix usdl-price: <http://www.linked-usdl.org/ns/usdl-pricing#> .
@prefix sarest: <http://www.knoesis.org/research/srl/standards/sa-rest/#> .
```

```
@prefix usdl-sla: <http://www.linked-usdl.org/ns/usdl-servicelevel#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix conf: <http://purl.org/ontology/conference#> .

#
# Substrate Description Starts
#

<http://www.csp2.com/substrates/77554389-935a-4f74-8a59-54fc9d410321/description/> rdf:type
subs:SubstrateDescription ;
   dcterms:title "Audio Mixer Substrate Description";
   dcterms:creator :CSP2 ;
   usdl:hasProvider :CSP2;
:CSP2 a cloud:SubstrateProvider ;
   gr:legalName "CSP2 Communications Inc." ;
   foaf:page <http://www.csp2.com/> ;
   org:address [ a org:PostalAddress ;
            org:streetAddress "1000 Rue Berri" ;
            org:postalCode "H7N 4G9" ;
            org:addressLocality "Montreal, Canada" ] ;
   org:telephone "+1(514)-537-5037" ;
   org:faxNumber "+1(514)-537-5038" ;
   org:email "contact@csp2.com" .
:S2 a subs:Substrate;
           usdl:exposes subs:AudioMixer ;
           usdl:hasProvider :CSP2 ;
           usdl:hasInteractionProtocol :AudioMixerInteractionProtocol ;
           usdl:hasServiceModel :AudioMixerSubstrateModel .
```

```
#
# Audio Mixer Substrate Interactions
#

:AudioMixerInteractionProtocol a usdl:InteractionProtocol;
     dcterms:title "Technical Interactions";
     dcterms:description "Details the basic set of operations available for this substrate.";
     usdl:hasInteraction :AudioMixerInterfaces .

#
# Audio Mixer Substrate REST-based Operations
#
:AudioMixerInterfaces a usdl:Interaction ;
     dcterms:title "REST-based Interactions" ;
     dcterms:description "End Point details for REST-based Interactions" ;
     usdl:hasInterfaceOperation :ReserveMix ;
     usdl:hasInterfaceOperation :GetMixInfo ;
     usdl:hasInterfaceOperation :RemoveMix ;
     usdl:hasInterfaceOperation :ReserveParticipantMix ;
     usdl:hasInterfaceOperation :GetParticipantMixInfo ;
     usdl:hasInterfaceOperation :StartStopParticipantMedia ;
     usdl:hasInterfaceOperation :DeleteParticipantMix ;
     usdl:hasInterfaceOperation :ChangeParticipantMix .

:GetParticipantMixInfo a sarest:Operation ;
          dcterms:title "Get a particular participant's mix info" ;
          dc:description "Get a particular participant's mix info, specifically current media session information." ;
          sarest:hasAddress "Mix/{MixId}/Participant/{ParticipantId}"^^sarest:URITemplate ;
          subs:supportsDataFormats "application/json, application/xml, text/plain"^^rdf:literal ;
          subs:hasRequestUriParameter <http://purl.org/ontology/conference#MixId> ;
          subs:hasRequestUriParameter <http://purl.org/ontology/conference#ParticipantId> ;
          subs:affectedResources <http://purl.org/ontology/conference#Mix> ;
```

```
            sarest:hasMethod "GET"^^sarest:HTTPMethod ;
            subs:hasResponseCode "200 OK"^^http:StatusCode ;
            subs:hasResponseBody <http://purl.org/ontology/conference#MediaDescriptor> .
:ReserveMix a sarest:Operation ;
            dcterms:title "Reserve Mix " ;
            dc:description "Register the Conference record in the substrate" ;
            sarest:hasAddress "Mix/" ;
            subs:supportsDataFormats "application/json";
            subs:supportsDataFormats "application/xml";
            subs:supportsDataFormats "text/plain";
            subs:hasRequestBodyParameter <http://purl.org/ontology/conference#Conference> ;
            sarest:hasMethod "POST"^^sarest:HTTPMethod ;
            subs:hasResponseCode "201 Created" .

:GetMixInfo a sarest:Operation ;
            dcterms:title "Get mix " ;
            dc:description "Gets the information about the Mix. This equivalent to a Conference with all the participant
information." ;
            sarest:hasAddress "Mix/{MixId}" ;
            subs:supportsDataFormats "application/json";
            subs:supportsDataFormats "application/xml";
            subs:supportsDataFormats "text/plain";
            subs:hasRequestUriParameter <http://purl.org/ontology/conference#MixId> ;
            sarest:hasMethod "GET"^^sarest:HTTPMethod ;
            subs:hasResponseCode "200 OK" ;
            subs:hasResponseBody <http://purl.org/ontology/conference#Conference> .

:RemoveMix a sarest:Operation ;
            dcterms:title "Removes mix" ;
            dc:description "Removes the information about the mix from the substrates" ;
            sarest:hasAddress "Mix/{MixId}" ;
            subs:supportsDataFormats "application/json";
```

```
        subs:supportsDataFormats "application/xml";
        subs:supportsDataFormats "text/plain";
        subs:hasRequestUriParameter <http://purl.org/ontology/conference#MixId> ;
        sarest:hasMethod "DELETE"^^sarest:HTTPMethod ;
        subs:hasResponseCode "200 OK" .

:ReserveParticipantMix a sarest:Operation ;
        dcterms:title "Create Participant " ;
        dc:description "Connect the Participant's Media Session" ;
        sarest:hasAddress "Mix/{MixId}/Participant/" ;
        subs:supportsDataFormats "application/json";
        subs:supportsDataFormats "application/xml";
        subs:supportsDataFormats "text/plain";
        subs:hasRequestBodyParameter <http://purl.org/ontology/conference#Participant> ;
        sarest:hasMethod "POST"^^sarest:HTTPMethod ;
        subs:hasResponseCode "201 Created" ;
        subs:hasResponseBody <http://purl.org/ontology/conference#MediaDescriptor> .

:GetParticipantMixInfo a sarest:Operation ;
        dcterms:title "Get a particular participant's mix" ;
        dc:description "Get a particular participant's mix info, specifically current media session information." ;
        sarest:hasAddress "Mix/{MixId}/Participant/{ParticipantId}" ;
        subs:supportsDataFormats "application/json";
        subs:supportsDataFormats "application/xml";
        subs:supportsDataFormats "text/plain";
        subs:hasRequestUriParameter <http://purl.org/ontology/conference#MixId> ;
        subs:hasRequestUriParameter <http://purl.org/ontology/conference#ParticipantId> ;
        sarest:hasMethod "GET"^^sarest:HTTPMethod ;
        subs:hasResponseCode "200 OK" ;
        subs:hasResponseBody <http://purl.org/ontology/conference#MediaDescriptor> .

:UpdateParticipantMedia a sarest:Operation ;
```

```
            dcterms:title "Start Participant media" ;
            dc:description "Start sending participant the mixed packets" ;
            sarest:hasAddress "Mix/{MixId}/Participant/{ParticipantId}/MediaDescriptor/{MediaDescriptorId}" ;
            subs:supportsDataFormats "application/json";
            subs:supportsDataFormats "application/xml";
            subs:supportsDataFormats "text/plain";
            subs:hasRequestUriParameter <http://purl.org/ontology/conference#MixId> ;
            subs:hasRequestUriParameter <http://purl.org/ontology/conference#ParticipantId> ;
            subs:hasRequestUriParameter <http://purl.org/ontology/conference#MediaDescriptorId> ;
            subs:hasRequestBodyParameter <http://purl.org/ontology/conference#MediaDescriptor> ;
            sarest:hasMethod "PUT"^^sarest:HTTPMethod ;
            subs:hasResponseCode "202 Accepted" ;
            subs:isAsynchronous "true"^^xsd:boolean ;
            subs:hasCallBackUri :CallBackUri ;
            subs:hasCallBackMethod :CallBackMethod ;
            subs:hasCallBackResponseCode :CallBackExpectedResponseCode ;
            subs:hasCallBackBodyParameter <http://purl.org/ontology/conference#MediaDescriptor> .

:DeleteParticipantMix a sarest:Operation ;
            dcterms:title "Delete Participant from a mix" ;
            dc:description "Delete the Participant's Media mix information" ;
            sarest:hasAddress "Mix/{MixId}/Participant/{ParticipantId}" ;
            subs:supportsDataFormats "application/json";
            subs:supportsDataFormats "application/xml";
            subs:supportsDataFormats "text/plain";
            subs:hasRequestUriParameter <http://purl.org/ontology/conference#MixId> ;
            subs:hasRequestUriParameter <http://purl.org/ontology/conference#ParticipantId> ;
            sarest:hasMethod "DELETE"^^sarest:HTTPMethod ;
            subs:hasResponseCode "200 OK" .
```

```
#
# Price Plan (Cost Information)
#
:CSP2StandardPlan a usdl-price:PricePlan;
   dcterms:title "Standard Plan";
   dcterms:description "Standard Plan that fits most business users." ;
   usdl-price:hasPriceCap [ a gr:PriceSpecification;
                     gr:hasCurrency "CAD";
                     gr:hasCurrencyValue "29.99"^^xsd:float;
                     gr:unitOfMeasurement "per/month" ];
   usdl-price:hasPriceFloor [ a gr:PriceSpecification;
                      gr:hasCurrency "CAD";
                      gr:hasValueFloat "9.99"^^xsd:float;
                      gr:unitOfMeasurement "per/month" ];
   usdl-price:hasPriceComponent [ a usdl-price:PriceComponent;
                        dcterms:title "Monthly rate";
                        dcterms:description "Monthly fee to pay for the substrate.";
                        gr:hasCurrency "CAD";
                        gr:hasCurrencyValue "0.25"^^xsd:float;
                        gr:unitOfMeasurement "per month per user"
                       ];
   usdl-price:hasTax           [ a usdl-price:PriceComponent;
                        dcterms:title "Total Tax";
                        dcterms:description "Provincial and Federal tax component";
                        gr:hasValueFloat "15.0"^^xsd:float;
                        gr:unitOfMeasurement "percent"
                       ] .
:AudioMixerSubstrateOffering a usdl:ServiceOffering;
   usdl:includes :S2;
   usdl:hasPricePlan :CSP2StandardPlan;
   usdl:validFrom "2012-05-19T19:08:24.798Z"^^xsd:dateTime;
   usdl:validThrough "2015-05-19T19:08:24.798Z"^^xsd:dateTime;
   sla:hasServiceLevelProfile :AudioMixerSubstrateProfile .


:AudioMixerSubstrateModel a usdl:ServiceModel;
       usdl:hasNature usdl:Automated;
       usdl:versionInfo "1.0";
       usdl:hasClassification [ a skos:ConceptScheme;
                                    skos:hasTopConcept subs:AudioMixer ;
                               rdfs:label "Audio Mixer"];
       dcterms:modified "2011-05-19T19:08:24.798Z"^^xsd:dateTime;
       dcterms:created "2011-05-19T19:08:24.798Z"^^xsd:dateTime;
       dcterms:title "Audio Mixer Solution";
       usdl:shortDescription "Audio Mixer Solution for Conferencing";
       usdl:longDescription "<b>CSP-2 Cloud-based Audio Mixer</b> Substrate for Conferencing" .
#
# Constraints
#
:AudioMixerSubstrateProfile a sla:ServiceLevelProfile;
      dcterms:title "Standard Service Profile";
      sla:hasServiceLevel [
           a sla:GuaranteedState;
           dcterms:title "Capacity";
           sla:serviceLevelExpression [
              a sla:ServiceLevelExpression;
              dcterms:description "<b>[MixingCapacity]</b> Maximum number for the mixes.";
              sla:hasVariable  :MixingCapacity, :BitrateProfileLow, :BitrateProfileMed, :BitrateProfileHigh ] ;
           sla:obligatedParty usdl:Provider
          ],
          [
           a sla:GuaranteedState;
           dcterms:title "Performance";
           sla:serviceLevelExpression [
              a sla:ServiceLevelExpression;
```

```
                dcterms:description "";
                sla:hasVariable  :ResponseTime, :Latency ] ;
            sla:obligatedParty usdl:Provider
           ],
           [ a sla:GuaranteedState;
            dcterms:title "Interoperability";
            sla:serviceLevelExpression [
                a sla:ServiceLevelExpression;
                dcterms:description "Codecs supported by the Mixer Substrate";
                sla:hasVariable  :MixerSupportedAudioCodecs ] ;
            sla:obligatedParty usdl:Provider
           ] .
:MixingCapacity
             a sla:Variable;
             rdfs:label "Maximum Mixing Capacity";
             sla:hasDefault [ a gr:QuantitativeValueInteger ;
                              gr:hasValue “160"^^xsd:integer ;
                              gr:unitOfMeasurement "users" ] .

:MixerSupportedAudioCodecs
       a sla:Variable;
             rdfs:label "Supported Audio Mixer Codecs";
             sla:hasDefault [ a gr:QualitativeValue ;
                              gr:hasValue "G711,Speex" ] .
:ResponseTime
             a sla:Variable;
             rdfs:label "ResponseTime";
             sla:hasDefault [ a gr:QuantitativeValueInteger;
                              gr:hasValue "2500"^^xsd:integer;
                              gr:unitOfMeasurement "milli seconds" ] .
:Latency
             a sla:Variable;
```

```
        rdfs:label "Latency";
        dcterms:description "Latency for Audio Mixer Substrate" ;
        sla:hasDefault [ a gr:QuantitativeValueInteger;
                    gr:hasValue "1000"^^xsd:integer;
                    gr:unitOfMeasurement "milli seconds" ] .
:BitrateProfileLow
        a sla:Variable;
        rdfs:label "BitrateProfileLow";
        sla:hasDefault [ a gr:QuantitativeValueInteger ;
                    gr:hasValue "64"^^xsd:integer ;
                    gr:unitOfMeasurement "kbps" ] .
:BitrateProfileMed
        a sla:Variable;
        rdfs:label "BitrateProfileMed";
        sla:hasDefault [ a gr:QuantitativeValueInteger ;
                    gr:hasValue "128"^^xsd:integer ;
                    gr:unitOfMeasurement "kbps" ] .

:BitrateProfileHigh
        a sla:Variable;
        rdfs:label "BitrateProfileHigh";
        sla:hasDefault [ a gr:QuantitativeValueInteger ;
                    gr:hasValue "384"^^xsd:integer ;
                    gr:unitOfMeasurement "kbps" ] .
```

Appendix B: Query (including SPARQL Ranking function) for discovering top 10 Audio Mixer substrates with Canadian region

```
#
# Considers three constraints for Ranking – Price, Latency, and Availability
# All values are normalized based on the entire SDs in semantic data store
#
# Availability is calculated on the fly using aggregate function on number of mirror properties for a substrates

SELECT DISTINCT ?prName ?region ?source ?subsf ?price ?nprice ?latency ?nlatency (COUNT(?mirror) AS ?availability)
?navailability ?capacity ?ncapacity
(subs:substrateRank(?nprice, ?nlatency, ?navailability, ?ncapacity, ?region, "132.205.164.209") AS ?rank) WHERE {
   {
    ?sd rdf:type subs:SubstrateDescription ;
     usdl:offers ?o ;
     <http://purl.org/dc/elements/1.1/source> ?source ;
     usdl:hasProvider ?provider .
?source subs:normalizedPrice ?nprice ;
  subs:normalizedLatency ?nlatency ;
  subs:normalizedAvailability ?navailability ;
  subs:normalizedCapacity ?ncapacity .
{
?provider gr:legalName ?prName ;
     <http://xmlns.com/foaf/0.1/page> ?website ;
     <http://schema.org/Organization#address> ?paddress ;
   ?paddress <http://schema.org/Organization#streetAddress> ?saddresss ;
      <http://schema.org/Organization#postalCode> ?postalCode ;
      <http://schema.org/Organization#addressLocality> ?region .
   FILTER regex(?region, "Canada") .
}
```

```
   ?o rdf:type usdl:ServiceOffering ;
      usdl-sla:hasServiceLevelProfile ?sf ;
     usdl:hasPricePlan ?pp ;
      usdl:includes ?subs .
   ?subs subs:mirrors ?mirror .
   ?subs usdl:exposes  subs:AudioMixer .
{
  ?sf rdf:type usdl-sla:ServiceLevelProfile ;
      usdl-sla:hasServiceLevel ?sl .
   ?sl <http://purl.org/dc/terms/title> ?slvtitle .
   FILTER regex(?slvtitle, "Performance") .
   ?sl rdf:type usdl-sla:GuaranteedState ;
      usdl-sla:serviceLevelExpression ?sle .
   ?sle rdf:type usdl-sla:ServiceLevelExpression ;
      usdl-sla:hasVariable ?slv .
   ?slv rdfs:label ?vlab .
   FILTER regex(?vlab, "Latency") .
   ?slv usdl-sla:hasDefault ?valueexpr .
   ?valueexpr gr:hasValue ?latency .
}
 {
         ?sf rdf:type usdl-sla:ServiceLevelProfile ;
           usdl-sla:hasServiceLevel ?sl2 .
         ?sl2 <http://purl.org/dc/terms/title> ?slvtitle2 .
         FILTER regex(?slvtitle2, "Capacity") .
         ?sl2 rdf:type usdl-sla:GuaranteedState ;
            usdl-sla:serviceLevelExpression ?sle2 .
         ?sle2 rdf:type usdl-sla:ServiceLevelExpression ;
            usdl-sla:hasVariable ?slv2 .
         ?slv2 rdfs:label ?vlab2 .
         FILTER regex(?vlab2, "Capacity") .
         ?slv2 usdl-sla:hasDefault ?valueexpr2 .
```

```
            ?valueexpr2 gr:hasValue ?capacity .
        }
{
?pp usdl-price:hasPriceCap ?pcap .
   ?pcap gr:hasCurrency "CAD" ;
         gr:hasCurrencyValue ?price .
}
OPTIONAL { ?sd rdfs:label ?slab . }
}}
GROUP BY ?prName ?region ?source ?subsf ?price ?latency ?nprice ?nlatency ?navailability ?ncapacity ?capacity
ORDER BY DESC (?rank)
LIMIT 10
```

Bibliography

[1] S. NIST, *800-145: The NIST definition of cloud computing*. 2012.

[2] J. Li, R. Guo, and X. Zhang, ‘Study on service-oriented Cloud conferencing’, in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, 2010, vol. 6, pp. 21–25.

[3] R. H. Glitho, ‘Cloud-based Multimedia Conferencing: Business Model, Research Agenda, State-of-the-Art’, in *2011 IEEE 13th Conference on Commerce and Enterprise Computing (CEC)*, 2011, pp. 226 –230.

[4] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, ‘A break in the clouds: towards a cloud definition’, *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2008.

[5] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, ‘Cloud computing — The business perspective’, *Decis. Support Syst.*, vol. 51, no. 1, pp. 176–189, Apr. 2011.

[6] Q. Zhang, L. Cheng, and R. Boutaba, ‘Cloud computing: state-of-the-art and research challenges’, *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, 2010.

[7] M. Barnes and C. Boulton, ‘A Framework for Centralized Conferencing - RFC 5239’, RFC 5239, June, Jun. 2008.

[8] G. Camarillo, K. Drage, and J. Ott, ‘Binary Floor Control Protocol (BFCP) - RFC 4582’, 2006.

[9] O. Novo, G. Camarillo, and D. Morgan, ‘Conference Information Data Model for Centralized Conferencing (XCON) - RFC 6501’, RFC 6501, March, 2012.

[10] ‘3GPP TS 24.147, Conferencing Using the IP Multimedia (IM) Core Network (CN), Stage 3, Release 11’, Mar-2008. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/24147.htm. [Accessed: 13-Jun-2013].

[11] ‘3GPP TS 29.199-12, Open Service Access (OSA), Parlay-X Web Services – Part 12: Multimedia Conference (Release 9)’. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/29199-12.htm. [Accessed: 13-Jun-2013].

[12] ‘RFC 4353 - A Framework for Conferencing with the Session Initiation Protocol (SIP)’. [Online]. Available: http://tools.ietf.org/html/rfc4353. [Accessed: 25-Jun-2013].

[13] ‘W3C Semantic Web Activity’. [Online]. Available: http://www.w3.org/2001/sw/. [Accessed: 28-Jun-2013].

[14] V. Sugumaran and J. A. Gulla, *Applied semantic web technologies*. Auerbach Pub, 2012.

[15] ‘Introduction to Semantic Web Technologies’, 02-Jun-2010. [Online]. Available: http://www.w3.org/2010/Talks/0622-SemTech-IH/. [Accessed: 30-May-2013].

[16] A. Maedche and S. Staab, ‘Ontology learning for the Semantic Web’, *IEEE Intell. Syst.*, vol. 16, no. 2, pp. 72 – 79, Apr. 2001.

[17] K. Breitman, M. A. Casanova, and W. Truszkowski, *Semantic web: concepts, technologies and applications*. Springer, 2007.

[18] G. Klyne, J. J. Carroll, and B. McBride, ‘Resource description framework (RDF): Concepts and abstract syntax’, *W3C Recomm.*, vol. 10, 2004.

[19] Manu Sporny, Gregg Kellogg, and Markus Lanthaler, ‘W3C JSON-LD Syntax 1.0’. [Online]. Available: https://dvcs.w3.org/hg/json-ld/raw-file/b53e28df4bfd/spec/WD/json-ld-syntax/20120712/index.html. [Accessed: 10-Jan-2013].

[20] D. L. McGuinness and F. Van Harmelen, ‘OWL web ontology language overview’, *W3C Recomm.*, vol. 10, no. 2004–03, p. 10, 2004.

[21] F. Manola, E. Miller, and B. McBride, ‘RDF Primer. W3C Recommendation (2004)’, *World Wide Web Consort. W3C Httpwww W3 OrgTR2004REC-Rdf-Prim.-20040210Last Access Date 30 Aug 2004*, 2004.

[22] ‘Resource Description Framework - Wikipedia, the free encyclopedia’. [Online]. Available: http://en.wikipedia.org/wiki/Resource_Description_Framework. [Accessed: 30-May-2013].

[23] K. Dentler, R. Cornet, A. ten Teije, and N. de Keizer, ‘Comparison of reasoners for large ontologies in the OWL 2 EL profile’, *Semantic Web*, vol. 2, no. 2, pp. 71–87, 2011.

[24] ‘SPARQL Query Language for RDF’. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/. [Accessed: 30-May-2013].

[25] ‘Web Services Architecture’. [Online]. Available: http://www.w3.org/TR/ws-arch/. [Accessed: 24-Jul-2013].

[26] N. Loutas, V. Peristeras, and K. Tarabanis, ‘Towards a reference service model for the Web of Services’, *Data Knowl. Eng.*, vol. 70, no. 9, pp. 753–774, 2011.

[27] D. Anicic, M. Brodie, J. de Bruijn, D. Fensel, T. Haselwanter, M. Hepp, S. Heymans, J. Hoffmann, M. Kerrigan, and J. Kopecky, ‘A semantically enabled service oriented architecture’, in *Web Intelligence Meets Brain Informatics*, Springer, 2007, pp. 367–381.

[28] D. Martin and J. Domingue, ‘Semantic web services, part 1’, *Intell. Syst. IEEE*, vol. 22, no. 5, 2007.

[29] J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell, ‘Sawsdl: Semantic annotations for wsdl and xml schema’, *Internet Comput. IEEE*, vol. 11, no. 6, pp. 60–67, 2007.

[30] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, and T. Payne, ‘OWL-S: Semantic markup for web services’, *W3C Memb. Submiss.*, vol. 22, pp. 2007–04, 2004.

[31] D. Fensel and C. Bussler, ‘The web service modeling framework WSMF’, *Electron. Commer. Res. Appl.*, vol. 1, no. 2, pp. 113–137, 2002.

[32] T. Vitvar, J. Kopecky, M. Zaremba, and D. Fensel, ‘WSMO-Lite: lightweight semantic descriptions for services on the web’, in *Fifth European Conference on Web Services, 2007. ECOWS ’07*, 2007, pp. 77 –86.

[33] E. Triantaphyllou, B. Shu, S. N. Sanchez, and T. Ray, ‘Multi-criteria decision making: an operations research approach’, *Encycl. Electr. Electron. Eng.*, vol. 15, pp. 175–186, 1998.

[34] J. R. S. C. Mateo, ‘Weighted Sum Method and Weighted Product Method’, in *Multi Criteria Analysis in the Renewable Energy Industry*, Springer London, 2012, pp. 19–22.

[35] H. H. Wang, N. Gibbins, T. Payne, A. Saleh, and J. Sun, ‘A Formal Semantic Model of the Semantic Web Service Ontology (WSMO)’, in *12th IEEE International Conference on Engineering Complex Computer Systems, 2007*, 2007, pp. 74 –86.

[36] ‘Linked USDL’. [Online]. Available: http://www.linked-usdl.org/. [Accessed: 30-Jun-2013].

[37] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, and T. Payne, ‘DAML-S: Web service description for the semantic web’, in *The Semantic Web—ISWC 2002*, Springer, 2002, pp. 348–363.

[38] D. L. McGuinness and F. Van Harmelen, ‘OWL Web Ontology Language - Overview’, *W3C Recomm.*, vol. 10, no. 2004–03, p. 10, 2004.

[39] ‘Workflow Management Coalition’. [Online]. Available: http://www.wfmc.org/. [Accessed: 20-Jul-2013].

[40] R. Lara, D. Roman, A. Polleres, and D. Fensel, ‘A conceptual comparison of WSMO and OWL-S’, in *Web services*, Springer, 2004, pp. 254–269.

[41] ‘D24.2v0.1. WSMO Grounding’. [Online]. Available: http://wsmo.org/TR/d24/d24.2/v0.1/. [Accessed: 16-Jun-2013].

[42] K. Kadner and D. Oberle, ‘Unified Service Description Language XG Final Report’. [Online]. Available: http://www.w3.org/2005/Incubator/usdl/XGR-usdl-20111027/. [Accessed: 27-May-2013].

[43] D. Brickley and L. Miller, ‘FOAF vocabulary specification 0.98’, *Namespace Doc.*, vol. 9, 2010.

[44] S. Weibel, ‘The Dublin Core: a simple content description model for electronic resources’, *Bull. Am. Soc. Inf. Sci. Technol.*, vol. 24, no. 1, pp. 9–11, 2005.

[45] M. Hepp, ‘Goodrelations: An ontology for describing products and services offers on the web’, *Knowl. Eng. Pr. Patterns*, pp. 329–346, 2008.

[46] C. Pedrinaci, D. Liu, M. Maleshkova, D. Lambert, J. Kopecky, and J. Domingue, ‘iServe: a linked services publishing platform’, in *CEUR Workshop Proceedings*, 2010, vol. 596.

[47] J. Kopecky, K. Gomadam, and T. Vitvar, ‘hRESTS: An HTML Microformat for Describing RESTful Web Services’, 2008, vol. 1, pp. 619 –625.

[48] M. Maleshkova, J. Kopecký, and C. Pedrinaci, ‘Adapting SAWSDL for Semantic Annotations of RESTful Services’, in *On the Move to Meaningful Internet Systems: OTM 2009 Workshops*, R. Meersman, P. Herrero, and T. Dillon, Eds. Springer Berlin Heidelberg, 2009, pp. 917–926.

[49] J. Li and F. Yang, ‘Resource-Oriented converged network service modeling’, in *Communications Technology and Applications, 2009. ICCTA’09. IEEE International Conference on*, 2009, pp. 895–899.

[50] A. N. Khan, S. Asghar, and S. Fong, ‘Framework of integrated Semantic Web Services and Ontology Development for Telecommunication Industry’, *J. Emerg. Technol. Web Intell.*, vol. 3, no. 2, pp. 110–119, 2011.

[51] D. Bernstein and D. Vij, ‘Using semantic web ontology for intercloud directories and exchanges’, *Proc. ICOMP*, vol. 10, 2010.

[52] N. Loutas, E. Kamateri, and K. Tarabanis, ‘A Semantic Interoperability Framework for Cloud Platform as a Service’, in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, 2011, pp. 280–287.

[53] S. K. Garg, S. Versteeg, and R. Buyya, ‘SMICloud: a framework for comparing and ranking cloud services’, in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, 2011, pp. 210–218.

[54] T. M. Smith, ‘Reusable features for VoIP service realization’, in *Principles, Systems and Applications of IP Telecommunications*, 2010, pp. 42–47.

[55] ‘OASIS UDDI Specification TC | OASIS’. [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec. [Accessed: 09-Jan-2013].

[56] D. Kourtesis and I. Paraskakis, ‘Combining SAWSDL, OWL-DL and UDDI for semantically enhanced web service discovery’, *Semantic Web Res. Appl.*, pp. 614–628, 2008.

[57] Y. Ait-Ameur, ‘A Semantic Repository for Adaptive Services’, in *2009 World Conference on Services - I*, 2009, pp. 211 –218.

[58] C. Wu and E. Chang, ‘Aligning with the web: an atom-based architecture for web services discovery’, *Serv. Oriented Comput. Appl.*, vol. 1, no. 2, pp. 97–116, 2007.

[59] V. X. Tran, H. Tsuji, and R. Masuda, ‘A new QoS ontology and its QoS-based ranking algorithm for Web services’, *Simul. Model. Pr. Theory*, vol. 17, no. 8, pp. 1378–1398, 2009.

[60] T. L. Saaty, ‘The analytical hierarchical process’, *J Wiley New York*, 1980.

[61] T. Haselwanter, P. Kotinurmi, M. Moran, T. Vitvar, and M. Zaremba, ‘WSMX: A Semantic Service Oriented Middleware for B2B Integration’, in *Service-Oriented Computing – ICSOC 2006*, A. Dan and W. Lamersdorf, Eds. Springer Berlin Heidelberg, 2006, pp. 477–483.

[62] J. De Bruijn, H. Lausen, A. Polleres, and D. Fensel, ‘Web Service Modeling Language WSML: An Overview’, in *The Semantic Web: Research and Applications*, Springer, 2006, pp. 590–604.

[63] I. Toma, T. Bürger, O. Shafiq, and D. Döegl, ‘GRISINO - An Integrated Infrastructure for Semantic Web Services, Grid Computing and Intelligent Objects’, in *The Semantic Web: Research and Applications*, S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, Eds. Springer Berlin Heidelberg, 2008, pp. 869–873.

[64] B. de hOra and J. Gregorio, ‘The Atom Publishing Protocol’, 2007.

[65] ‘Serializing SPARQL Query Results in JSON’. [Online]. Available: http://www.w3.org/TR/rdf-sparql-json-res/. [Accessed: 18-Mar-2013].

[66] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, ‘Importing the semantic web in UDDI’, *Web Serv. E-Bus. Semantic Web*, pp. 815–821, 2002.

[67] A. Duke, S. Stincic, J. Davies, G. Álvaro Rey, C. Pedrinaci, M. Maleshkova, J. Domingue, D. Liu, F. Lecue, and N. Mehandjiev, ‘Telecommunication mashups using RESTful services’, *Serv.-Based Internet*, pp. 124–135, 2010.

[68] Jos de Bruijn and Dieter Fensel, ‘Web Service Modeling Language (WSML) - W3C Submission’. [Online]. Available: http://www.w3.org/Submission/WSML/. [Accessed: 18-Jan-2013].

[69] N. Baklouti, B. Gargouri, and M. Jmaiel, ‘An ontology-based approach for Linguistic Web Service description’, in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on*, 2012, pp. 450–455.

[70] ‘XML Schema Datatypes in RDF and OWL’. [Online]. Available: http://www.w3.org/TR/swbp-xsch-datatypes/. [Accessed: 02-Jul-2013].

[71] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds, ‘SPARQL basic graph pattern optimization using selectivity estimation’, in *Proceedings of the 17th international conference on World Wide Web*, 2008, pp. 595–604.

[72] R. Castillo, C. Rothe, and U. Leser, ‘RDFMatView: Indexing RDF Data using Materialized SPARQL queries’, in *The 6th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2010)*, 2010, p. 80.

[73] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, ‘QoS-aware middleware for web services composition’, *Softw. Eng. IEEE Trans.*, vol. 30, no. 5, pp. 311–327, 2004.

[74] ‘Sesame Framework’. [Online]. Available: http://www.openrdf.org/index.jsp. [Accessed: 13-Jul-2013].

[75] ‘Jersey Web Service Framework’. [Online]. Available: https://jersey.java.net/. [Accessed: 23-Jun-2013].

[76] ‘Apache Jena - Reasoners and rule engines: Jena inference support’. [Online]. Available: http://jena.apache.org/documentation/inference/. [Accessed: 23-Jun-2013].

[77] ‘Apache Any23 Library’. [Online]. Available: http://any23.apache.org/. [Accessed: 12-Jul-2013].

[78] ‘RDF2Go Library for RDF Store Abstraction’. [Online]. Available: http://semanticweb.org/wiki/RDF2Go. [Accessed: 12-Jul-2013].

[79] 'Apache Tika Library for Content Analysis'. [Online]. Available: http://tika.apache.org/. [Accessed: 12-Jul-2013].

[80] 'RDF2Go - semanticweb.org'. [Online]. Available: http://semanticweb.org/wiki/RDF2Go. [Accessed: 26-Jun-2013].

[81] 'Berlin SPARQL Benchmark'. [Online]. Available: http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/. [Accessed: 12-Jul-2013].