Master’s thesis

Master of Science in Engineering – Computer Engineering, 120 higher education credits

Semantic Web mechanisms in Cloud Environment

Saeed Haddadi Makhsous
Mid Sweden University
The Department of Information Technology and Media (ITM)

Author: Saeed Haddadi Makhsous
E-mail address: saeid.special@gmail.com
Study programme: Master of Science in Engineering - Computer Engineering, 120 higher education credits
Examiner: Professor Tingting Zhang, Tingting.Zhang@miun.se
Academic supervisor: Dr. Ulf Jennehag , Ulf.Jennehag@miun.se
Supervisor at SAP AG: Dr. Martin Knechtel, Martin.Knechtel@sap.com
Scope: 14157 words
Date: 2014-07-22
Abstract

Virtual Private Ontology Server (VPOS) is a middleware software with focus on ontologies (semantic models). VPOS is offering a smart way to its users how to access relevant part of ontology dependent on their context. The user context can be expertise level or level of experience or job position in a hierarchy structure. Instead of having numerous numbers of ontologies associated to different user contexts, VPOS keeps only one ontology but offers sub-ontologies to users on the basis of their context. VPOS also supports reasoning to infer new consequences out of assertions stated in the ontology. These consequences are also visible for certain contexts which have access to enough assertions inside ontology to be able to deduct them. There are some issues within current implementation of VPOS. The application uses the random-access memory of local machine for loading the ontology which could be the cause of scalability issue when ontology size exceeds memory space. Also assuming that each user of VPOS holds her own instance of application it might result into maintainability issues such as inconsistency between ontologies of different users and waste of computational resources. This thesis project is about to find some practical solutions to solve the issues of current implementation, first by upgrading the architecture of application using new framework to address scalability issue and then moving to cloud addressing maintainability issues. The final production of this thesis project would be Cloud-VPOS which is an application made to deal with semantic web mechanisms and function over cloud platform. Cloud-VPOS would be an application where semantic web meets cloud computing by employing semantic web mechanisms as cloud services.

Keywords: Semantic web, Semantic data store, Ontology, SAP HANA cloud, SAP HANA database, Apache Jena framework
Acknowledgements

Thanks to:

Professor Tingting Zhang, Mid Sweden University, for her comments ensuring the thesis project meets all scientific criteria.

Dr. Ulf Jennehag, Mid Sweden University, for his supervision and advices regarding making a well-structured report.

Dr. Martin Knechtel, SAP AG, for his precise supervision and administration during this project.

Dr. Wei Wei, SAP AG, A cool computer guy who loves music.
# Table of Contents

Abstract ........................................................................................................................................ ii  
Acknowledgements ................................................................................................................... iii  
Table of Contents ...................................................................................................................... iv  
Terminology ..................................................................................................................................... vi  

1 Introduction ........................................................................................................................................ 1  
1.1 Background and problem motivation ...................................................................................... 2  
1.2 Overall aim ................................................................................................................................. 3  
1.3 Scope ........................................................................................................................................... 3  
1.4 Concrete and verifiable goals ....................................................................................................... 4  
1.5 Outline ......................................................................................................................................... 4  

2 Related work ....................................................................................................................................... 6  
2.1 Semantic Web ............................................................................................................................... 6  
2.1.1 Data .......................................................................................................................................... 6  
2.1.2 Ontologies ............................................................................................................................... 10  
2.1.3 Inference .................................................................................................................................. 11  
2.2 Semantic data store ....................................................................................................................... 12  
2.3 Context dependent view to Semantic Web Ontologies ............................................................... 14  
2.3.1 Technical preliminaries ........................................................................................................... 14  
2.3.2 Running example ..................................................................................................................... 16  
2.3.3 Computing a Boundary .......................................................................................................... 17  
2.4 Application design for the Cloud ............................................................................................... 18  
2.4.1 SAP HANA Cloud Platform .................................................................................................. 19  
2.4.2 SAP HANA Database ............................................................................................................. 21  

3 Methodology .................................................................................................................................... 23  
3.1 Virtual Private Ontology Server ............................................................................................... 23  
3.1.1 Overview of current architecture of VPOS ............................................................................. 24  
3.1.2 Problems within current implementation of VPOS .............................................................. 25  
3.2 Cloud-Based VPOS ..................................................................................................................... 26  
3.2.1 Motivation ............................................................................................................................... 26  
3.2.2 Architecture ........................................................................................................................... 27  
3.3 Appropriate framework for re-implementing VPOS ................................................................. 28  
3.4 Convenient architecture and platform for Cloud-VPOS ......................................................... 29  

4 Implementation ............................................................................................................................... 32
Semantic Web mechanisms in Cloud Environment
Saeed Haddadi Makhsous

Table of Contents

4.1 Design goals ................................................................. 32
4.2 VPOS Architecture using Jena Framework ......................... 32
4.2.1 Using Jena RDF API .................................................. 33
4.2.2 Using Jena Inference API .......................................... 34
4.2.3 Using Jena Store API ................................................ 35
4.3 Cloud Deployment .......................................................... 36
4.3.1 Cloud-VPOS Server .................................................. 37
4.3.2 Cloud-VPOS Client ................................................... 41
4.3.3 HANA Database Server ............................................. 41
4.4 Case study: car manufacture ontology ............................... 42

5 Results .............................................................................. 45
5.1 Advantages of cloud-based ontology server ......................... 45
5.2 Evaluation of the implementation ..................................... 46
5.2.1 Inferred consequences ............................................... 46
5.2.2 Hierarchy of users ..................................................... 49

6 Conclusions ...................................................................... 52
6.1 Re-implementing of VPOS with new framework ................... 52
6.2 Semantic data store as central data storage ......................... 52
6.3 Correcting issues about scalability and usability/maintainability .............................................................................. 53
6.4 Compromising semantic mechanisms with cloud environment .............................................................................. 53
6.5 Case study of Car Manufactory .......................................... 53
6.6 Web interface for Cloud-VPOS .......................................... 54
6.7 Possibility of including inference to results ......................... 54
6.8 Ethics perspective ............................................................ 54
6.9 Future work ..................................................................... 54

References ............................................................................. 56
Terminology

Acronyms:

VPOS Virtual Private Ontology Server
W3C World Wide Web Consortium
RDF Resource Description Framework
RDFS Resource Description Framework Schema
OWL Web Ontology Language
RIF Rule Interchange Format
XML Extensible Markup Language
URI Uniform Resource Identifier
DBMS Database Management System
SAS Statement on Auditing Standards
ISAE International Standard on Assurance Engagements
1 Introduction

In the recent years the World Wide Web has considerably changed. The vast growing amount of data and enormous number of users accessing web has made it very difficult to find, access, and maintain relevant information. This is because information is presented in natural languages which are not understandable for machines. Another reason is about volume of information which makes data management almost impossible without infrastructures aimed for this purpose.

In response to these problems many research have been initiated to enrich the web by adding semantic to the information. The goal of these researches was to “bring the web to its full potential”. As a result of collaboration between research consortiums and many commercial enterprises the Semantic Web was born as future of the current web. Semantic Web is the next generation of World Wide Web which allows future applications to understand knowledge and communicate knowledge base. Semantic Web leads Web to its full potential enabling machines to read information and automating services to empower the web beyond its current capabilities. Automated services capacity will improve by “understanding” the content of the web which is assisting human to achieve their desired information. Using semantic mechanisms these services are able to suggest more accurate methods for searching, filtering, and categorizing of information resources. [1]

On the other side to manage challenging amount of data Cloud Computing deliver computing resources over Internet. Cloud computing provides on-demand network access to a shared pool of configurable computing resources including servers, networks, data storage space, computer processing power, and user applications and services. It allows individuals and businesses to access hardware and software located on remote machines with minimal management efforts. The cloud computing model allows access to computer resources and cloud services from anywhere which has access to network. [2]

Semantic Web is already an established technology which needs more time for transferring all scientific ideas into a widely used technology while cloud computing is a fully implemented functioning technology. This thesis makes a bridge between semantic web and cloud computing
by providing a practical solution for comprising two technologies into an application which capable of presenting semantic mechanisms as cloud services.

1.1 Background and problem motivation

The proposed thesis will be conducted while the author is working in the EU FP7 project ebbits [3]. “The ebbits project does research in architecture, technologies and processes, which allow businesses to semantically integrate the Internet of Things into mainstream enterprise systems and support interoperable end-to-end business applications. It will provide semantic resolution to the Internet of Things and hence present a new bridge between backend enterprise applications, people, services and the physical world” [4].

The focus of this thesis project is on Virtual Private Ontology Server (VPOS), a component of the ebbits software platform. The semantic knowledge about the relevant concepts of the application domain is formulated in a structured format, more specifically ontology. The ebbits project and particularly VPOS need these ontologies to provide interoperable end-to-end business applications.

VPOS uses a centralized storage to keep all ontologies while labelling them for authorization purposes. Its function is to restrict reading access to a given labelled ontology. VPOS offers sub-ontologies of a large ontology as views to users. These sub-ontologies are presented to users based on contexts like level of details associated to job role, the appropriate level of information relevant to user proficiency, or the level of trust chosen for each user. Instead of keeping a separate ontology for each user context VPOS keeps only one ontology with possibility of providing sub-ontologies as views to relevant contexts. VPOS also utilizes the reasoners implemented in Description Logics (DL) [5] systems to derive implicit consequences from explicitly described knowledge in the ontologies.

The increasing amount of data available in the Internet has challenged the scalability for the storage systems. One of the main concerns in the ebbits project is scalability, since its software platform should be operable in different scales. VPOS utilizes File System to store and retrieve ontologies. Its design is suitable for medium data volume and operating
procedures containing query and reasoning are performed fully in main memory. Scalability is the main concern in the current implementation of VPOS, since it is unable to function on ontology files exceeding memory space.

Another issue is about maintainability, since ebbits component using VPOS application has to host the same ontologies as all others do, even though a major part of the ontologies may not be in any business interest of the owner of the component. This does not only result in resource wastes, but also leads to security and maintenance issues.

1.2 Overall aim

The project aim is to re-implement VPOS is such a way to address issues within current implementation. The framework for re-implementation is cloud platform as goal of this project is to comprise semantic web mechanisms with cloud computing in order to transfer VPOS into Cloud-VPOS capable of presenting semantic mechanisms as cloud services.

This project goal is to facilitate accessing VPOS services over internet and cloud and solving issues regarding scalability and maintainability. The users of Cloud-VPOS are no longer concerned about semantic models (ontologies) size and version since Cloud-VPOS assures semantic models are updated and they are available in any scale. In the new architecture of Cloud-VPOS all semantic models are moved to semantic data store as a central data storage enabling users to benefit Cloud-VPOS services without wasting any computational storage for keeping semantic models locally. While semantic data store is capable of hosting semantic data models in any possible size assuring the scalability.

Cloud-VPOS provides two main options for their users enabling them to choose if they want to access their sub-ontology including inferred consequences or not. Cloud-VPOS calculates inferred consequences of ontology and by using labelling mechanisms it decides which of them are accessible for the context of user. And then it waits to see if user wants these inferred consequences in addition to her semantic model or not.

1.3 Scope

As the name of this project describes it the focus of this project is on cloud computing and more specifically semantic web. The main concern
of this project in semantic web is about semantic models and offering some solutions in order to store, maintain, and manage them. While semantic models or ontologies are indeed the core focus of Cloud-VPOS the application itself is developed on cloud platform which can be defined as framework of this project. In this project it is important to not only offer users parts of semantic models (sub-ontology) relevant to their context but also extra results inferred implicitly.

Accessing results over internet as cloud services is also a necessity in this implementation. To be more specific this project is about to solve issues within current implementation of VPOS and provide a cloud compatible version of VPOS capable of presenting sub-ontologies to user over network. Another significant criteria to be considered in this project is semantic data store which is a prerequisite for implementation of Cloud-VPOS.

1.4 **Concrete and verifiable goals**

The questions which will be answered in this report can be listed as follows:

- i. What is Virtual Private Ontology Server (VPOS) and how it functions?
- ii. What are the issues within current implementation of VPOS?
- iii. How Cloud based implementation of VPOS (Cloud-VPOS) could address these issues?
- iv. What is the appropriate framework for re-implementing of VPOS?
- v. What is the convenient platform for Cloud-VPOS?
- vi. How to benefit Cloud-VPOS services on cloud environment?

1.5 **Outline**

Chapter 2 goes through theoretical concepts necessary for implementation of this project mostly regarding semantic web and cloud computing. In this chapter also related works are covered which provide a clear picture regarding context dependent views of ontology and how the early implementation of VPOS functions. Chapter 3 talks about
methods used to answer problems stated in section 1.4 as concrete goals. This chapter explains solutions which have been utilized in this project containing step by step plan for addressing issues within current implementation of VPOS and how to move to cloud. Chapter 4 discusses the implementation phase of project and explains the deployment procedures have been made to transform VPOS to Cloud-VPOS. The Cloud-VPOS is production of implementing procedures at different levels including re-implementing VPOS with new ontology framework, establishing semantic data store, and deploying the application into cloud. Chapter 5 takes a closer look at results and Chapter 6 declares conclusions of this thesis project.
2 Related work

2.1 Semantic Web

“The Semantic Web is not a separate web but an extension to the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation” [6]. The Semantic Web is considered as evolution of documents readable just for human to information understandable for computers. The web pages are full of tags and information which are surrounded by them but the meaning is absent in this syntax. Within current structure of the Web technology, computers have no understanding about meaningful contents of the web pages. They can just simply parse information and layouts of the web pages to present them on the screen.

Semantic means meaning which can provide more meaningful utilization of underlying layers of data in the web’s contents. The Semantic Web is designed to enable computers to manipulate information within the Web pages meaningfully. On the Semantic Web data is interpreted by machines, allowing them to perform the work involved in finding, combining, and acting upon information on the Internet.

Semantic Web has five main elements: Data, Ontology, Query, Inference, and Applications which together build fundamental of this technology. In the following section we talk about main elements of Semantic Web and its architecture.

2.1.1 Data

“The Semantic Web is a Web of Data — of dates and titles and part numbers and chemical properties and any other data one might conceive of. The collection of Semantic Web technologies (RDF, OWL, SPARQL, etc.) provides an environment where application can query that data, draw inferences using vocabularies, etc”. [7]

Linked Data

To put Semantic Web into practise, having huge amount of data in the standard format is vital. This data should be reachable and manageable
by semantic tools. Not only data is necessary but relationships among data are important and should be available. The collection of interrelated data can be also referred as Linked Data.

Based on World Wide Web Consortium (W3C) [7] Linked Data is the heart of Semantic Web. The Semantic Web focuses on integration of data in large scales and reasoning on data. Almost all applications written in the Semantic Web rely on accessibility and integration of Linked Data.

Resource Description Framework (RDF)
The Resource Description Framework (RDF) is a framework for representing data in the Semantic Web. Meaning in the Semantic Web is expressed by RDF which is encoded in collections of triples. The triples are written in Extensible Markup Language (XML). Each triple consists of three components: Subject, Predicate, and Object. In RDF each entity (Subject) has some properties (Predicate) with certain values (Object). In a triple the relationship between two things is indicated by some simple facts which are presented as Predicate, and the Subject and Object are the two things. Each component of triple is identified by Universal Resource Identifier (URI). Using a specific URI for each specific concept helps to distinguish the concept uniquely. A set of RDF triples produce a RDF Graph.

As an example consider Figure 2.1. Figure 2.1 is showing the representation for a person called Eric Miller. The RDF uses URIs to identify [8]:

- individuals, e.g., Eric Miller, identified by http://www.w3.org/People/EM/contact#me
- kinds of things, e.g., Person, identified by http://www.w3.org/2000/10/swap/pim/contact#Person
- properties of those things, e.g., mailbox, identified by http://www.w3.org/2000/10/swap/pim/contact#mailbox
- values of those properties, e.g. mailto:em@w3.org as the value of the mailbox property
RDF Schema
RDF Schema is a set of classes with certain properties which enables people to write their own RDF vocabulary. RDF Schema is layered on top of RDF using RDF extensible knowledge presentation to provide basic elements for describing ontologies. RDF Schema uses the same data model used in object-oriented programming languages like Java in such a way that it allows creating classes of data. In RDF Schema data model a class is described as a group of things with common characteristic while in object-oriented programming a class is a template of an object comprises of characteristics (members) and behaviors (methods). Both data models also allow for inheritance which enables classes to inherit characteristics from super classes and also producing a hierarchical structure among classes.

In the Figure 2.2 the layered approach of “Semantic Web Stack” is demonstrated. The concepts of universal identification (URI) and universal character set (Unicode) make the base of stack. On top of those concepts XML syntax is placed offering elements, attributes, angle
brackets, and namespaces as well to avoid vocabulary conflicts. Above XML is RDF as the primary representation language for semantic web including triple-based assertions as discussed in the previous section. When triples are used for denoting class, class property, and value, it can produce class hierarchies in order to classify and describe objects. This is the goal of RDF Schema, as mentioned earlier.

Above RDF Schema there are ontologies which are discussed in the next chapter. RDF Schema can be employed for writing taxonomy or lightweight ontologies while more detailed ontologies can be produced using Web Ontology Language OWL. OWL is a language based on description logics which offers more constructs compare to RDFS. It is syntactically related to RDF and like RDFS it provides additional standardized vocabularies for creating more advanced ontologies.

SPARQL is a SQL-like language using a simple protocol to query RDF Data as well as RDFS and OWL ontologies. SPARQL is able to query knowledge based ontologies denoting RDF triple and resources for both matching part and also returning part of the query. Logic Rules are about things and resources inside ontologies. A rule language is used to infer new knowledge from existing knowledge based ontologies. Additionally rules can provide solutions for filtering data during query procedure. If we consider rules as “introductory logic” the Logic layer will be “advance logic” allowing formal proofs to be shared. Formal proofs with trusted input for the proofs establish the trust layer showing “web of trust”. Cryptography and Digital signatures are used to verify the origin of data sources, supporting the web of trust.
2.1.2 Ontologies

Linked Data and RDF are not already enough for shaping the structure of the Semantic Web. The programs which want to compare or combine information across different data sets should be able to detect different identifiers for the same concept. There would be a problem when two data sets use different terms for referring to the same concept on the Web. A solution to this problem is provided by another basic component of the Semantic Web, called Ontology. Ontology is a collection of information in the format of file or document which formally defines the relationships between terms.

The main role of Ontologies on the Semantic Web is to help data integration when ambitious terms used in different data sets. Another role is about extra knowledge which may lead into extracting new relationship between terms. As an example you can consider applications working with ontologies in the health care. Medical professional use them to present the knowledge about diseases, symptoms, and treatments.
Pharmaceutical use them to represent information about drugs and dosages. Combining the knowledge from medical and pharmaceutical communities with patient’s data enables a range of intelligent applications such as “Decision support tools” to search for possible treatments or support different research areas in the medical science.

Ontologies can be considered as a solution to the terminology problem. The meaning of terms in a Web page can be referred to an Ontology page which is written in XML codes and defines the context of those associated terms. Even though using ontologies seems to be a proper way to avoid terminology problem, it’s not yet enough. When different ontologies contains different term for the same concept and web pages are associated to those ontologies the problem still exists. In this case there should be equivalence relationship between terms in different ontologies providing meaning for the same concept.

An example for this could be a bookseller who wants to integrate information from different publishers. The data is imported into publisher’s databases in common RDF model. However one database might use “Author” while the other one uses “Creator”. In order to complete data integration extra information should be added to RDF model expressing that terms “Author” and “Creator” are the same.

### 2.1.3 Inference

According to World Wide Web Consortium (W3C) [10] inference on the Semantic Web is about discovering new associations. On the Semantic Web, the data is formed as a set of associations between resources. “Inference” refers to the procedure of generating new associations between resources based on the data and based on some additional facts in the form of rules. These new associations can be explicitly added to the data or returned as additional implicit knowledge at query time.

Extracting this extra information on the Semantic Web is done via ontologies and rule sets which are drawn upon knowledge representation techniques. Ontologies in one hand concentrate on the methods for classification with emphasizing on definition of ‘classes’ and ‘subclasses’ and how individual resources can be associated to these classes. They also characterize the relationships between classes and their instances. On the other hand, the concentration is on Rules which are describing general mechanisms for learning and creating new relationships using
existing ones, similar to logic programs, like Prolog. In the Semantic Web family recommended by W3C the RDFS and OWL are means for defining ontologies while RIF (Rule Interchange Format) has been made for supporting rule based approaches.

**What Inference is used for?**

“Inference on the Semantic Web is one of the tools of choice to improve the quality of data integration on the Web, by discovering new relationships, automatically analyzing the content of the data, or managing knowledge on the Web in general. Inference based techniques are also important in discovering possible inconsistencies in the (integrated) data”. [10]

**Examples**

A simple example might be helpful. Supposing the data set considered for this example contains the relationship (Victoria isA Cat). Another ontology may state that “Cat is also a Pet” which means that a Semantic Web application capable of understanding the concept of “X is also Y” can insert the statement (Victoria isA Pet) to the data set even though it wasn’t part of original data. In another word a new relationship was “discovered”. One more example can be about the fact that “when two persons have the same name, email address, and home page they are identical”. In this case inferencing can be used to discover the “identity” of two resources. [10]

### 2.2 Semantic data store

To handle and store Semantic Data, stated inside ontologies, a persistent, efficient, and flexible database technology is necessary. The database should be able to manage large datasets and perform efficient data access and queries for accessing resources and individuals and their properties when they match some certain circumstances.

The underlying data model of ontology to be stored in database is RDF. Ontology data is corresponded to RDF triples which all together form a complex RDF graph. RDF graphs can be stored in particular databases, called RDF triple stores, abbreviated as triple stores. Triple store utilize the query language SPARQL for managing and querying the large
graphs and they typically comprise a query framework and underlying backend.

Generally triple stores can be divided into three types: Native stores which rely on native database engine as persistent storage system, optimized for RDF processing. DBMS-backed stores which use a relational database management system as backend to present the RDF Model in relational schema. The third type is Hybrid stores, which support both architectures. Triple stores are able to handle very large datasets even more than one billion triples, with continues grows for achieving better performance and optimization. Currently OWLIM-Enterprise is the largest known triple store capable of handling 20 billion triples on a single server.

Database representation of RDF
Available RDF triple stores use different Storage schemas for storing triples. The most popular database representation for striping RDF/S resources into relational databases is the Schema-Oblivious (also called generic or vertical). There are other schemas such as the Schema-Aware (also called specific or binary) and a hybrid representation which combines the features of other two. The difference among these three representations is about the definition and usage of different table design for storing RDF triples. The schema-oblivious approach employ only one table for holding all triples (the triples table, see Figure 2.3), while the schema-aware uses separate table for each class and for each property. The hybrid approach provides one table for all class definition memberships and one table for each property instance with its different range value. [11]

<table>
<thead>
<tr>
<th>Subject (resource URI)</th>
<th>Property (property URI)</th>
<th>Object (Literal value or resource URI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 2.3: The schema-oblivious database layout [11]

The open source framework Jena SDB [12], which is used in the implementation of this project, is built upon schema-oblivious approach. This approach simplifies the extension of triple store when there are further classes and properties to be handled, without the need for creating or deleting tables, which is required in the schema-aware approach.
Triple store management
Since Jena SDB as underlying triple store employ schema-oblivious approach, it utilizes only one database table for storing all RDF triples. This means that all ontology data represented as RDF triples are stored in the same table. The management of triple store generally comprises two main operations: adding and deleting of triples. Adding operation simply includes loading the ontology files, parsing them into triples, and sending all triples to the database. Since every resource or individual stored in triple store owns a unique URI, the deleting operation contains deleting triples from triple store which the subject or object URI matches the specific URI of deleted item.

2.3 Context dependent view to Semantic Web Ontologies
Here we are going to provide a framework for offering sub-ontologies to users as views base on their contexts like access right of user, the level of details needed by user, or the trust level required by application. As a replace of having different sub-ontologies for different contexts, this solution propose to keep only one ontology but equip each axiom inside the ontology with the appropriate label from context lattice where different context of this ontology are expressed by this lattice.

Employing reasoning procedures for large ontologies some certain consequences like subsumption are pre-computed. Instead of pre-computing these consequences for each context separately, they can be computed once but be labeled for further access by different context. In this approach one label (called boundary) is computed for each consequence in a way that a comparison between user label and consequence label determines if the consequence follow from the sub-ontology associated with user context. This approach has nothing to do with the internal of reasoning procedure but use these procedures directly as sub-procedure without any modification. The research is also limited about knowledge presentation in the format of Description Logic axiom. Axioms are statements or rules stated in ontology which are accepted true without being proved.

2.3.1 Technical preliminaries
Assume that there is an ontology O which we want to offer different views on this ontology to different users based on their context. In
another word each user can only see a subset of ontology defined by the context the user operating in. The context might be the access right granted to user, the level of experience she has or the details seems to be appropriate for the current setting, etc. To be more concrete, context dependent view can be used to reduce the load of information by providing only appropriate data to the experience level of user. For example in a medical ontology one offer can be about patient which has only basic knowledge, one for general doctor, one for an urologist, etc. Another example could be about commercial ontologies where some certain policies define access level. Within these policies the context of each user is evaluated by her assigned roles and then the availability of some axioms and their implicit consequences for that context is decided.

The idea is to have just one large ontology $O$ but assign “labels” to the axioms inside ontology and to the users such that a comparison of the axiom label and user label determine if the axiom belongs to the sub-ontology of user or not. The comparison is computationally cheap and can be implemented efficiently using an index structure to look up all axioms for a given label.

“To be more precise, we use a set of labels $L$ together with a partial order $\leq$ on $L$ and assume that every axiom $\alpha \in O$ has an assigned label $\text{lab}(\alpha) \in L$. The partial order of $(L, \leq)$ should define a lattice which is stronger restriction for $L$. The labels $l \in L$ are also used to define user contexts (which can be interpreted as access rights, required level of experience, etc.). The sub-ontology accessible for the context with label $l \in L$ is defined to be $O \geq l := \{\alpha \in O | \text{lab}(\alpha) \geq l\}$. [13]

Obviously, users not only should have access to their axioms but also to the consequences derived from those axioms. This means that a user whose context is $l$ should be able to see all consequences of $O \geq l$. The solution to this problem is to compute a so-called boundary for the consequence $c$, i.e., an element $v$ of $L$ such that $c$ follows from $O \geq l$ if and only if $l \leq v$ [13]. Then instead of pre-computing whether each consequence follows from every possible sub-ontology, this approach computes only one label for every consequence in a way that a comparison between the context label and the consequence label determine if the consequence is valid for the corresponding sub-ontology or not.
In order to stay as general as possible and avoid full details and proofs a running example is provided containing enough details to understand the main idea behind this approach.

2.3.2 Running example

The scenario for this running example is about access restriction. The scenario assumes semantically annotated documents describing Web services are offered on a Web marketplace, like traditional goods are sold on eBay, Amazon, etc. Different types of users are assigned different type of permission to enable them to create, sell, buy, and advertise the services. Access is not only restricted to individual documents but also to a large ontology including all the semantic annotation in one place.

Consider an ontology O from a marketplace in the Semantic Web representing knowledge about the Ecological Value Calculator service (eco-Calc), EU Ecological Services (EUecoS), High Performance Services (HPerfS), services with few customers (SFewCust), services generating low profit (LowProfitS), and services with a price increase (SPrIncr) having the following axioms [13]:

\begin{align*}
\alpha_1: & \text{EUecoS } \sqcap \text{HPerfS(ecoCalc)} \\
\alpha_2: & \text{HPerfS } \sqsubseteq \text{SFewCust } \sqcap \text{LowProfitS} \\
\alpha_3: & \text{EUecoS } \sqsubseteq \text{SFewCust } \sqcap \text{LowProfitS} \\
\alpha_4: & \text{SFewCust } \sqsubseteq \text{SPrIncr} \\
\alpha_5: & \text{LowProfitS } \sqsubseteq \text{SPrIncr}
\end{align*}

“The assertion SPrIncr(ecoCalc) is a consequence of O that follows from each of the minimal axiom sets \{\alpha_1, \alpha_2, \alpha_4\}, \{\alpha_1, \alpha_2, \alpha_5\}, \{\alpha_1, \alpha_3, \alpha_4\}, and \{\alpha_1, \alpha_3, \alpha_5\}, and has three diagnoses (sets of axioms you need to remove from O so that the consequence does not follow anymore), namely \{\alpha_1\}, \{\alpha_2, \alpha_3\}, and \{\alpha_4, \alpha_5\”. [13]

Here an explanation is provided about how lattices can be used to encode contexts and help to solve reasoning problems related to them. For this running example the goal is to produce an access list to regulate authorized permissions for each user based on his user role. The focus in this example is on reading access. A typical representation of user roles
and their permission for accessing the objects is the Access Control Matrix (Lampson 1971). The lattice for this running example is obtained using access control matrix, depicted in Figure 2.4.

The lattice itself is the set of elements \{l_0, l_1, l_2, l_3, l_4, l_5\} represented by circles and the lines connecting them presenting the order between them. Some of the elements are special in the sense that they are assigned with the context of users, i.e. user roles. These elements are \{l_0, l_2, l_3, l_5\}. The assignment of axioms is described below.

Assuming \((L, \leq)\) be applicable to the lattice shown in the Figure 2.4, while elements \(l_0, l_2, l_3, l_5\) represent different type of users (which are, contexts) that have access to an ontology. Suppose ‘lab’ be the labeling function to assign each axiom \(\alpha_i\) of ontology \(O\) from example the label \(l_i\) as shown in the Figure 2.4. The visible part for label \(l_3\) defining development engineer context is the sub-ontology \(O_{l\geq l_3} = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}\), with all consequences aside. Consider that labels which are lower in the lattice define larger context view to ontology. More particularly a user assigned a context lower in the lattice would have larger sub-ontology as well more number of axioms are visible to her (and thus, consequences) compare to a user belonging to a context above.

2.3.3 Computing a Boundary

As every axiom is visible only for certain contexts, an implicit consequence of the ontology is also inferable for those contexts which have access to enough axioms in order to deduce it. Computing boundary is
about computing suitable labels for such implicit consequences which express, just as the labels of the axioms, contexts capable of deducing them based on their visible axioms.

The journal article (Baader, Knechtel, Peñaloza, 2012) presents the technical notions, algorithms and proofs their correctness in detail. To clarify the basic principle we continue focusing on the running example where each axiom \( a_i \) is labeled with \( \text{lab}(a_i) = l_i \). To compute the boundary for a consequence we should first specify the minimal axiom sets which the consequence follows from. The existing sets for the consequence \( \text{SPRlnr}(\text{ecoCalc}) \) are \{\( a_1, a_2, a_4 \)\}, \{\( a_1, a_2, a_5 \)\}, \{\( a_1, a_3, a_4 \)\}, and \{\( a_1, a_3, a_5 \)\}. Next step is to compute the infimum of the axiom’s labels for each minimal axiom set which respectively are \( l_3, l_0, l_3, \) and \( l_0 \). The suprimum of those 4 axiom set labels is boundary which is \( l_3 \oplus l_0 \oplus l_3 \oplus l_0 = l_3 \). The boundary particularly shows that the contexts associated to labels \( l_3 \) and \( l_0 \), respectively, development engineers and customer service employees, are able to drive and see the consequence. [13]

2.4 Application design for the Cloud

Application design for the cloud follows a tired approach. Using this approach the application is deployed in three separate layers; Interface layer, Application layer, and Database layer. The goal of this design is to make application loosely coupled and provider independent. This approach offers flexibility of deploying each layer for separate service provider if it was necessary to take advantage of provider strengths. Also layer design of application helps to speed up development and allows for using more standard frameworks. Figures 3.3 demonstrate the general architecture of cloud computing and application design for cloud platform.

Section 2.4.1 introduces Cloud Platform used in this project for application deployment while section 2.4.2 covers SAP HANA database which is employed as semantic data storage.
2.4.1 SAP HANA Cloud Platform

SAP HANA Cloud offers modular “platform as a service” as an Eclipse-based open standard. The cloud platform is operated by SAP [15], and benefits latest industry advances in cloud standards. SAP HANA Cloud Platform applications can be deployed via OSGi frameworks or via command line tools as WAR files. The applications run within the Java-based runtime environment powered by SAP HANA and can be maintained via Web-based management tools. [16]

In order to maximize productivity in development, SAP HANA Cloud Platform leverages all well-known open-source tools and frameworks.
According to SAP HANA Cloud Platform datasheet [16] at this time SAP HANA Cloud Platform supports Java EE 6 Web Profile and a subset of the Java 7 SE specification. The open nature of the platform allows developers to work with other frameworks such as Spring Roo and other third-party. There is also the possibility of programming with other languages that compile to Java, including Scala, JRuby, Clojure, and Groovy. Support for other runtime environments, open-source tools, and frameworks is still in progress which enable developers to build cloud based applications using wide range of technologies.

Based on SAP HANA Cloud Platform datasheet [16] applications which are natively developed for SAP HANA Platform support representation state transfer (REST) services and Web services interfaces enabling them to integrate other cloud based applications. This characteristic allows developers to create loosely coupled applications that extend the value with respect to SAP or non-SAP enterprise solutions.

SAP HANA Cloud Platform is powered by persistence which is provided by SAP HANA, exploiting in-memory computing technology, real-time, and built-in analytics. SAP HANA is available for developers as a service which provides them the environment development without being worry about setting up or administrating of virtual database instances of SAP HANA.

SAP HANA Cloud Platform offers comprehensive, multilevel security measures to protect critical business data and to provide essential industry-standard compliance certifications. The protection comprises different levels including strong physical security of data centers, safeguarding the data, and ensuring full reliability of the services. SAP HANA Cloud Platform meets standards, including ISO 27001, Statement on Auditing Standards (SAS) 70 Type II, and International Standard on Assurance Engagements (ISAE) 3402 to ensure inclusive infrastructure security, stability, and performance. The certificates are always being renewed to afford reliable guard against evolving security threats. [16]

SAP HANA Cloud Platform is designed to provide a modular lightweight runtime environment for application deployment which includes a container (lean Java server) on a virtual machine. The platform feeds the applications with services provided centrally but shared logically across the platform to maximize the performance and reduce the overhead.
SAP HANA Cloud Platform provides a secure, scalable runtime environment offering the following reusable platform services [16]:

- "Persistency service that leverages the speed of SAP HANA"
- Connectivity service that enables secure integration with on-premise systems running software from SAP and other vendors
- Identity management service enabling single sign-on with third-party identity providers
- Scalable document service for managing unstructured content
- Mail service for processing outbound e-mail from applications
- SAP HANA Cloud Portal for mashing up OpenSocial gadgets from third-party providers with applications built on SAP HANA Cloud Platform"

### 2.4.2 SAP HANA Database

Traditional database management systems are designed to optimize the performance on hardware access with limited main memory. The main bottleneck was Disk I/O and most of the optimization was focused on disk access, for example to minimize the number of disk pages should be transferred to main memory during processing. The SAP HANA database foundation from beginning was based on the idea that main memory is available in a very large amount, considering the fact that memory size can theoretically reach 18 billion gigabytes or 18 exabytes for 64 bits systems, and no further I/O constraint for disk accessing. Instead of optimizing I/O hard disk access, SAP HANA database optimizes reading data between main memory and CPU cache. SAP HANA is an extremely parallel data management system which functioning fully in the main memory, providing possibility of row and column based storage, and supporting built-in multi-tenancy.

SAP HANA serves new generation of analytic and transactional applications developing fully in-memory. New applications developed natively for SAP HANA and in-memory process can increase the performance of business process and analytical scenarios noticeably. Application development techniques can take advantage of parallel in-memory processing
by utilizing new hardware technologies, optimized enterprise data management system and application development logic.

The table 2.1 reviews the benefits offered by specific features of the SAP HANA database.

<table>
<thead>
<tr>
<th>SAP HANA® Database Feature</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multicore CPU</td>
<td>Greater computation power</td>
</tr>
<tr>
<td>Large memory footprint</td>
<td>Faster than disk</td>
</tr>
<tr>
<td>Row and column store</td>
<td>Faster aggregation with column store</td>
</tr>
<tr>
<td>Compression</td>
<td>Highly dependent on actual data used</td>
</tr>
<tr>
<td>Partitioning</td>
<td>Analysis of large data sets</td>
</tr>
<tr>
<td></td>
<td>Complex computations</td>
</tr>
<tr>
<td>No aggregate tables</td>
<td>Flexible modeling</td>
</tr>
<tr>
<td>Nonmaterialized views</td>
<td>No data duplication</td>
</tr>
<tr>
<td>Insert only on delta</td>
<td>Fast data loads</td>
</tr>
</tbody>
</table>

Table 2.1: SAP HANA Database specific features [17]

The SAP HANA uses multi-core architecture for managing data and distributes data across all cores to maximize RAM locality using scale-out (horizontally) and scale-up (vertically) functionality. In the scale-out method, the SAP HANA database can be extended to a range of servers in one cluster rather than a single server. Large table are distributed across several servers using hash, round robin, or range partitioning. SAP HANA has this capability to execute queries across multiple servers and retain distributed transaction safely. [17]

One of the major arguments about traditional DBMS is about update procedure which data is being locked during this time. This feature results in lower performance in traditional DBMS while SAP HANA avoids this issue by enabling high level of parallelization using insert-only data records. Instead of inserting data as new records in a database table the data is inserted as net-new entries into existing records stored in columns.
3 Methodology

In this section, methods utilized to fulfill project goals are explained. The methods solve the problems mentioned in the part 1.4.

3.1 Virtual Private Ontology Server

Virtual Private Ontology Server is a middleware component of ebbits software platform which enables efficient extraction/query of sub-ontology form a large ontology based on user roles and their access rights. Instead of materializing a large ontology into different sub-ontologies, each axiom of the ontology is assigned a label with user role which has the right to access it. VPOS also pre-computes a label for each consequence of ontology which represents the set of user roles which are allowed to access the consequence. The associated labels of consequences are re-computed if there are relevant alterations in ontology or the access rights of users. This way, VPOS avoids the unnecessary duplication of semantic data while achieving higher performance for reasoning on large ontologies.

VPOS uses a centralized storage to keep all ontologies while labelling them for authorization purposes. Its function is to restrict reading access to a given labelled ontology. VPOS purpose to provide sub-ontology of a large ontology as views to the users according to their contexts. Instead of dividing a large ontology into different sub-ontologies, VPOS suggests to keep only the large ontology while assigning “labels” to the axioms of ontology and to the users in such a way that a simple comparison between user labels and axiom labels can determine if the user is allowed to access the relevant axioms or not.

VPOS calculates the visible part of ontology for each user according to the label already assigned to her. The assignment of label is done using appropriate context lattice and on the basis of user context which can be the access rights of a user, the level of details requested by the user, or the trust level needed by the application.
Computing a Boundary
VPOS utilizes the reasoners implemented in Description Logics (DL) systems to derive implicit consequences from explicitly described knowledge in ontologies. “Just as every axiom is accessible only for certain contexts, a consequence of the ontology will only be derivable in those contexts that have access to enough axioms to deduce it. VPOS computes adequate labels (called boundaries) for such implicit consequences, which express, just as the labels of the axioms, which contexts are capable of deducing them from their visible axioms” [13].

VPOS performs three main tasks on a labeled ontology:

1. Retrieving a sub-ontology defined by a label
2. Computing all consequences
3. Computing the label (boundary) for each consequence

3.1.1 Overview of current architecture of VPOS
As it is demonstrated in the Figure 3.1 VPOS uses OWL API [18] as Ontology API to read and exchange semantic data organized in the ontology format. The OWL API is a standardized programming interface for creating, processing and manipulating of OWL [19] Ontologies.

VPOS uses Reasoner as a Black-box that accepts ontology as input and a specific entailment to examine if the entailment is deductible from the ontology. VPOS uses reasoner as a sub-routine and does nothing about internals of the reasoner.

To store and retrieve ontologies VPOS uses File System. It means that ontologies are available in File System and Ontology API simply reads the ontology files. VPOS utilizes a central storage to obtain ontology files and they can be retrieved locally or remotely.
3.1.2 Problems within current implementation of VPOS

Within current implementation, VPOS is deployed in each ebbits component and ontologies are loaded from local files. Every ebbits component keeps the same copy of ontologies. There are two major drawbacks about VPOS implementation.

Scalability

The increasing amount of data available in the Internet has challenged the scalability for the storage systems. One of the main concerns in the VPOS implementation is scalability, since it should be operable in different scales. VPOS utilizes File System to store and retrieve ontologies. Its design is suitable for medium data volume and operating procedures containing query and reasoning are performed fully in main memory. Scalability is the main concern in the current implementation of VPOS, since it is unable to function on ontology files exceeding memory space.

The dependency of VPOS to file system prevents it to be operable on different scales. VPOS uses Ontology API to read the entire ontology file at once and keep it in memory for further steps. Ontology files might be available in different sizes and when the memory space (RAM) is not enough to hold the whole ontology file the application stops working. A better solution would be to store ontologies on semantic stores and retrieve data when needed.
Usability/Maintainability

Each user of ebbits using VPOS should keep the same ontologies as all others do. Since every user runs VPOS locally and VPOS should read ontology files in order to function correctly, it simply means that each user should separately assign computational hardware for storing ontology files. This doesn’t only result into waste of computational hardware but also into security and privacy issues. Because each user can access the whole ontology even though a major part of it is not even in her interest.

Inconsistencies between local copies of ontologies could be another issue. Whenever there is a change to ontology (probably small) the local copies of all other ebbits users should be updated. If the procedure of synchronization does not happen timely users might face different versions of ontologies. A better solution could be moving all ontologies into semantic data store as central storage and users only request to access semantic data necessary for their context.

3.2 Cloud-Based VPOS

Here we are going to provide an argument regarding Cloud-based VPOS and how it can address issues within current implementation of VPOS. Moreover in this section we provide a general picture describing the structure of system.

3.2.1 Motivation

The idea of implementing VPOS on the cloud environment is about providing a solution to meet issues within current implementation. In the cloud implementation, semantic models are stored and retrieved in the triple store which removes dependency to the file system. Replacing file system with triple store enables Cloud-VPOS to operate in different scales since it’s not dependent to the memory space of local machine any more. Embedding trip store as part of system structure enhances VPOS as a scalable application no matter which semantic model or ontology size the application works with.

The cloud-based VPOS is hosted by a central server which eliminates the necessity of holding VPOS by individual ebbits components. Each client can send request and receive results based on her business interest while avoiding the issues about keeping all ontologies and security and priva-
cy issues as well. Any update on semantic model happens on the central storage where clients can receive the newest version of semantic models not being worry about consistency issues.

### 3.2.2 Architecture

The cloud-based ontology server no longer needs individual ebbits components to hold VPOS. Instead it is hosted by a central server while each user just has a thin client for sending requests and receiving the results. The triple store can be placed at the same server as the ontology server or in a separate database server. By using triple store the understanding of Cloud-VPOS from semantic data is changing to triple or statement rather than axiom. Figure 3.2 displays the architecture of cloud-based ontology server.

At the server side, the ontology server employs semantic store API in order to store ontologies into database as triple store, establish the connection and query semantic data. As an alternative a triple store can be used directly. Inference API is utilized for reasoning, and RDF API is used for retrieving triples and presenting other ontology entities. For this implementation Apache Jena Framework is used which offers RDF API, Inference API, and Store API. The server side can be hosted in any cloud platform, which is this implementation the chosen cloud platform is SAP HANA Cloud. As mentioned earlier the database (or triple store) can be set up in a different server than ontology server.

Client side consists of a thin client for sending queries and demonstrating results. It doesn’t need to carry the database; it should only equip the user with the possibility questioning the visible parts of semantic model associated to the user role. The client side can be deployed as a java application or a mobile app for mobile devices or as a web page using web developing technologies.
Due to the need for storing ontologies in semantic data storage, VPOS should be re-implemented with a new framework capable of connecting to triple store. As a result of investigating possible options Apache Jena Framework was chosen as it offers a complete stack of APIs for developing each layer of ontology server.

Choosing Apache Jena as framework for deploying Cloud-VPOS was based on considering every requirement of project. Cloud-VPOS needs to manipulate RDF triples, entail additional facts, and store Semantic Data in Triple-store. Jena includes clearly defined subsystems for re-implementing different divisions of VPOS. RDF triples and graphs are accessed and retrieved using Jena’s RDF API. Inference can be done using Jena’s Inference API which provides a number of different rule engines for accomplishing this job. Jena’s Store API provides variety of different storage system such as in-memory store, SQL databases as triple store, or its proprietary native tuple store.

Jena stores semantic data as triples in directed graphs which allows adding, removing, retrieving, and manipulating of data using its major subsystem. Considering triple as the basic unit of semantic data enables
Cloud-VPOS to use a wide range of triple stores for ontologies. This feature provides possibility of easy plug in of different triple stores as far as they support Java API, which is a big step in generalization of Cloud-VPOS and generating a compatible implementation with different storage frameworks. In the Figure 3.2 you can see VPOS architecture using Jena framework comprising RDF API, Inference API, and Store API.

3.4 **Convenient architecture and platform for Cloud-VPOS**

Thinking about convenient structure for Cloud-VPOS led us to the application design suitable for cloud platform. This design follows a tired approach for application architecture. Within this approach the Cloud-VPOS would be implemented in three separate layers; Interface layer, Application layer, and Database layer.

Cloud deployment of VPOS needs to be done over Platform as a Service (PaaS) since it is targeted for application deployment. The cloud services provided on PaaS are programming environments (PE) and execution environments (EE) where Cloud-VPOS application written in Java programming language can be executed.

Figure 3.3 demonstrates Cloud-VPOS implementation following layered approach. Cloud implementation consists of User interface, VPOS re-implemented using Jena framework, and Semantic data store.
Interface Layer
The interface layer consists of suitable web page enabling users to communicate with Cloud-VPOS. The web page which is considered as user interface for Cloud-VPOS provides the possibility of choosing relevant context based on user’s role. The user should be able to see the sub-ontology and inferred consequences as response to her request.

Application Layer
The application layer comprises VPOS re-implemented using Jena framework. The re-implemented application is able to connect to semantic data store and retrieve the visible part of ontology for each user.
Cloud-VPOS enables users to choose if they want to access the sub-ontology including inferred result or not. Cloud-VPOS is capable of calculating the sub-ontology for each user for both scenarios. In the first scenario the user simply demands her sub-ontology relevant to her context while in the second scenario the user demands her sub-ontology and inferred consequences associated to sub-ontology.

**Database Layer**

The database layer includes semantic data store which is located in a separate database server. The separation of database server offers flexibility for taking advantage of a different service provider than provider which hosts the application. To use semantic data store, first semantic data model (ontology) should be loaded which needs using appropriate API for connecting to semantic data store and loading data. After loading data the semantic data store is ready to use.
4 Implementation

4.1 Design goals

Due to design goals defined for this project it should be able to manage these requirements:

- Central management and storage of Ontologies
- Central update of Semantic Models
- Remote access of services
- Providing access to sub-ontology based on user's context
- Deriving additional RDF assertions which are entailed from basic ontology
- Returning sub-ontology to the end user with or without inferred consequences

The implementation includes two main phases. At the first step the VPOS should be deployed using Jena framework. This phase solve the scalability issue of VPOS because it makes VPOS independent from memory of local machine. In the next step VPOS would be deployed into Cloud environment and it becomes Cloud-VPOS.

4.2 VPOS Architecture using Jena Framework

The necessity of replacing file system in the old implementation of VPOS with semantic data store is compelling new deployment of VPOS with new Ontology API capable of connecting to semantic data store. As mention in methodology chapter Apache Jena Framework was chosen as new framework due to its comprehensive stack of API meeting all requirements of Cloud-VPOS for managing semantic models (ontologies). Jena contains different APIs for developing each layer of application and the fact that these APIs are from same framework makes them compatible and easy to work.

Jena framework treat Semantic Data as triples and benefits perfectly rich RDF API. The priority about Jena is the store API and how it simplifies using relational or in-memory databases as underlying data store for semantic data. Replacing file system with triple store helps VPOS to
Semantic Web mechanisms in Cloud Environment
Saeed Haddadi Makhsous

4 Implementation
2014-07-22

avoid limitations of memory size and scalability concerns. Figure 4.1 demonstrates VPOS structure using Jena Framework:

![VPOS structure using Jena framework](image)

In the following sections the mechanisms used to fulfil the application requirements are explained in details.

4.2.1 Using Jena RDF API

Jena RDF API is a Java API which provides high-level programming interfaces to manipulate RDF resources. As a result of using Jena RDF API in the Cloud-VPOS the Semantic Data is stored and retrieved in the format of Triple or Statement which grant the privilege of using triple store to the application.

Earlier implementation of VPOS uses OWL API as Ontology API to manage semantic models written in Web Ontology Language (OWL). OWL API treats semantic data as axioms, while Jena RDF API considers triple as the basic unit of semantic data. The re-implementation of VPOS needs ontologies written in RDFS language.
Jena RDF API uses Model Interface [20] to create an RDF Model which is a set of statements (triples). Model Interface provides methods for adding, removing and accessing statements. Here you can see how Jena RDF API can access different parts of statement:

```java
// list the statements in the Model
StmtIterator iter = Model.listStatements();

// accessing predicate, subject and object of each statement
while (iter.hasNext()) {
    Statement stmt = iter.nextStatement();  // get next statement
    Resource subject = stmt.getSubject();     // get the subject
    Property predicate = stmt.getPredicate();  // get the predicate
    RDFNode object = stmt.getObject();        // get the object
}
```

Accessing different parts of statement enables Cloud-VPOS to retrieve certain statements from triple store associated to user’s context. In the new implementation the ontology models are written in RDFS language and a label is assigned to each RDF resource from ontology. Calculation of visible part of ontology for each user is determined based on comparing user’s label and labels already assigned to each RDF resource.

### 4.2.2 Using Jena Inference API

When it is necessary to derive additional assertions from explicit Semantic Data the Jena Reasoner is used. Jena Inference system allows using a range of different inference engines. The Jena Reasoner supports use of languages such as RDFS and OWL which allow additional facts to be inferred from instance data. Cloud-VPOS utilizes Jena Reasoner to entail implicit data out of stored triples in the Triple-Store.

In the implementation of VPOS the FaCT++ [21] was the only reasoner used for inference and the fact that it could only work with Axiom structure has made many limitations for adapting new APIs. Using Jena reasoner brings more compatibility to the Cloud-VPOS, since the entailment is based on Triple structure. Cloud-VPOS uses Jena RDFS reasoner with transitive specification. The focus is on relations with rdfs:subClassOf or rdfs:subPropertyOf as property value for the predicates.
4.2.3 Using Jena Store API

Cloud-VPOS uses Jena SDB (persistent triple stores using relational databases) in order to store and retrieve Semantic Data. SDB is a component of Jena framework used for RDF storage. SDB uses SQL databases as triple store and it supports variety of databases, OPEN Source or proprietary. Jena SDB enables Cloud-VPOS to utilize different relational databases as triple store.

SAP HANA database is the chosen triple store in the Cloud-VPOS. The privilege of supporting different SQL databases by Jena SDB makes the Cloud-VPOS compatible with different Cloud Platform. Since the destined cloud platform for Cloud-VPOS is SAP HANA Cloud the choice of SAP HANA Database as triple store becomes very obvious as the matter of compatibility and performance.

For semantic data to be accessible and manageable by Cloud-VPOS two steps should be taken; first loading of Ontology Models into triple store using Jena SDB scripts, and then accessing the data within Cloud-VPOS using Jena SDB Java API.

Loading ontology models into triple store

SDB store is accessible and manageable with command line scripts and via SDB Java API. The scripts are bash scripts which also run over Cygwin [22]. To copy ontology models previously stored in plain text file into triple store, first SDB environment variables should be set. These variables include SDBROOT as the root of SDB installation, database username and password for JDBC, and path to JDBC.

In Microsoft Windows the setup can be done using Cygwin.

```
$ export SDBROOT="/path/to/sdb"
$ export SDB_USER="YourDbUserName"
$ export SDB_PASSWORD="YourDbPassword"
$ export SDB_JDBC="/path/to/driver.jar"
$ export PATH=$SDBROOT/bin:$PATH
```

Then sdb.ttl file which is located in SDBROOT should be customized for chosen triple store (HANA Database). After finalizing customization the following commands will format HANA Database as a triple store and copy the Ontology files into storage.
4.3 Cloud Deployment

Up to now VPOS has been implemented with the new architecture using Apache Jena Framework. The new implementation makes VPOS capable of using triple store while keeping all features of labelling and context view of ontology models. The possibility of embedding SQL databases as triple store presents cloud compatible version of VPOS. Here in the Figure 4.2 the cloud implementation of VPOS is displayed. As you can see the server side is implemented on SAP HANA Cloud PaaS while a separate HANA Database Server hosts ontology models.

Before deploying VPOS into cloud, the ontology models should be loaded into HANA Database Server. Database should be ready in advance, loaded with semantic data and accessible online. This step is considered as pre-requirement while in the next step the VPOS is deployed on SAP HANA Cloud Platform which becomes Cloud-VPOS.

Figure 4.2: Cloud-VPOS Architecture
4.3.1 Cloud-VPOS Server

Since Cloud-VPOS should be deployed on HANA Cloud PaaS it should be written as a ‘Web Application’ allowing end-users to submit and retrieve data to/from triple store over the Internet using web browser. To be more technically oriented Cloud-VPOS as a web application query the database server and dynamically generate web documents for serving end-users. The generated document is in JSP (JavaServer Pages) which is a standard format of web technology permitting users to surf web interface of Cloud-VPOS in different browsers.

The key structure to implement Cloud-VPOS as Web Application is to add Servlet into its architecture. Since Cloud-VPOS should serve its users remotely as cloud service it should define HTTP-specific servlet using Java Servlet technology. The javax.servlet.http [23] package provides interfaces and classes for writing servlet. Extending HttpServlet class in Cloud-VPOS provides life-cycle methods, such as doGet and doPost, for handling HTTP-specific services.

```java
public class VPOS extends HttpServlet {
    public VPOS() {
        super();
        // TODO Auto-generated constructor stub
    }

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws
            ServletException, IOException {
        // TODO Auto-generated method stub
    }

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response) throws
            ServletException, IOException {
        // TODO Auto-generated method stub
    }
}
```

Data flow diagram

Considering the architecture of Cloud-VPOS shown in the Figure 4.2 following Data Flow Diagram explains the sequence of data submitted and retrieved to/from Cloud-VPOS by end-users. As it is shown in the figure 4.3, Cloud-VPOS has the following Data Flow:
1. The end user sends the HTTP-Request to access Web Interface of Cloud-VPOS (Surfing the web interface of application using web browser).

2. Since the Web Interface of Cloud-VPOS is a static web page (doesn’t contain any dynamic contents), the Web Server returns HTTP-Response directly to browser without requesting the application server. The HTTP-Response contains the Web Interface of Cloud-VPOS which can be seen in the figure 4.4.

3. The end user is now able to request for sub-ontology relevant to her role with or without inferred consequences. The request is submitted to Web Server.

4. Since the request is for dynamic resources the Web Server forwards it to the Application Server.

5. The Application Server passes the request to the correct web application (Cloud-VPOS).

   For user’s request not containing inference Cloud-VPOS only calculates Sub-Ontology defined by user label while it performs following steps in the case of user’s request for inferred consequences:

   a. retrieving the sub-ontology defined by user label
   b. computing all consequences
   c. computing the label for each consequence
   d. adding additional consequences with equal or greater labels to the sub-ontology

7. The response is passed back up to the chain to the Web Client.

8. The end user would observe her visible part of ontology containing RDF resources as well as triples which are relevant to the user role. The result can be seen in the figure 4.5.

![Figure 4.4: Cloud-VPOS Web Interface](image-url)
Deployment into cloud

The chosen platform for deployment of Cloud-VPOS is SAP HANA Cloud Platform. The deployment procedure contains following steps [24]:

1. Sign up for an SAP HANA Cloud account
   - Signing up for a SAP Developer Account offers access to the HANA Cloud with no time limitation and free of charge.

2. Set up Eclipse IDE with SAP HANA Cloud tools
   - ‘SAP HANA Cloud Platform SDK’ should be downloaded and installed. The location of installation directory should be set inside Eclipse
   - ‘SAP Development Tools’ should be downloaded and installed as a toolkit inside Eclipse

3. Create a Dynamic Web Project including Servlet
   - In order to deploy Cloud-VPOS into Cloud environment it should be created as a Web Application. To create a Web Application inside Eclipse IDE ‘Dynamic Web Project’ should be chosen as project type.

4. Deploying the project into the Cloud using Eclipse IDE
   - Setting up the SAP HANA Server as Runtime Environment inside Eclipse IDE
   - The Cloud-VPOS runs on this server and deployment would be completed
4.3.2 Cloud-VPOS Client

Client side (User Interface) is written as a Web Page using HTML and JSP technologies (Figure 4.4). The web page was design considering Cloud-VPOS functions and purposes. End users require accessing visible part of ontology based on their functionality or context. To observe preferred results they need to choose appropriate entry relevant to their role or context. The entry is taken by a `<form>` HTML tag which triggers the action of calling Cloud-VPOS on application server using post method.

```html
<form action="http://ApplicationServerURL:Port/CloudVPOS/" method="POST">
    ...
</form>
```

As you can see in the Figure 4.4 the entry contains user role and her willing for observation of inferred consequences. It is submitted to the Cloud-VPOS in Application Server and is handled by the post method of HttpServlet.

4.3.3 HANA Database Server

Deployment of HANA Database is done using a separate server on Amazon Web Services. A server instance is created on Amazon Elastic Compute Cloud [25] (Amazon EC2) which is hosting SAP HANA Database. Using Amazon EC2 allows obtaining and configuring capacity with minimal friction. It provides the complete control of HANA database and let’s running on Amazon proven computing environment.

Prerequisites for creating an instance of SAP HANA on AWS are:

- SAP Community Network (SCN) user account
- Amazon Web Service (AWS) user account
- Key pair

Amazon EC2 uses public-key cryptography to encrypt and decrypt login information. The procedure of creating key pair is simple and can be done inside Amazon EC2 console. When creation procedure completed the private key file automatically would be downloaded by the browser.
Once prerequisites were met, the SAP Developer License Agreement needs to be accepted on SCN and it will redirect you to the AWS website for creating SAP HANA instance. To continue with configuration of instance the size of the instance and the key pair name are important entries while the rest of details can be kept as default values.

Here is the instance made for this project hosting the HANA Database Server.

![Instance](image)

**Figure 4.6: SAP HANA Database Server on AWS**

To load Ontology Models into HANA Database Server the procedure explained in the section 4.2.3 should be followed. After finalizing data loading procedure the HANA Database Server is available and ready to use inside Cloud-VPOS. Accessing the database inside Cloud-VPOS is done using Jena SDB Java API.

### 4.4 Case study: car manufacture ontology

Due to the focus of ebbits project to the car manufactory the case study which is chosen for Cloud-VPOS is about car manufacturing. The case study offers role based context view in such a way that different users have different views to the car manufactory ontology. In another word each user can only see a subset of ontology defined by the context the user operating in.

The ontology for this case study consists of resources about different parts of car like engine, door, sensor, sunroof, gauge, etc. These parts are
categorized into 4 main groups which are Body, Electrical, Power, and Miscellaneous. In the Car Manufactory ontology we want to offer views to different technicians responsible for different parts of car. One view could be for Body technician who needs to access related parts of car like doors and windows and etc. One other view could be about Electrical technician, one for Power technician, one for Miscellaneous technician, etc. The following Figure demonstrates a portion of ontology graph for car manufactory case study.

Figure 4.7: Car manufactory ontology graph

In this case study every users can access the visible portion of ontology based on their area of expertise. The user can be “Body Technician”, “Electrical Technician”, “Power Technician”, “Miscellaneous Technician”, and “Auto Technician”. The semantic lattice for user roles is shown in the Figure 4.8 which also denotes the hierarchy of users. In this lattice Auto Technician is a user who has some general skills and can see only limited number of car parts from car manufactory ontology. Helper Node is not actually a user role but as its name explains it is helping to keep the lattice in partial order. You can see Body Technician below Helper Node which is considered as a specialist in the body related parts of car. So users assigned to Body role are able to observe all parts from ontology related to their role (parts of ontology with Body label assigned) and also parts related to the roles with less expertise like Auto Technician (parts of ontology which are assigned labels ≥ Body label). In this lattice we also assume that Nobody is expert in all divisions of car (Nobody have all the answers) and this label is not assigned to any RDF resources from car manufactory ontology.
Figure 4.8: Car manufactory lattice for user roles
5 Results

This chapter presents an overview of the implementation made in this thesis project. The project aimed to implement semantic web mechanisms over cloud infrastructure enabling users to access ontology using web and internet. Cloud-VPOS as final production of this project allows users to access the sub-ontology associated to their context using the web interface of application. There are several advantages of cloud-based ontology server which are reviewed in the following section.

5.1 Advantages of cloud-based ontology server

Central data storage
In the new architecture semantic data store (triple store) is used to store and manage data centrally which simply means that ontologies are no longer stored locally. Having central data storage enables individual ebbits components to access ontologies remotely which not only saves precious resources but also helps to improve efficiency, privacy, and security. Users don’t waist computational storage anymore to store a large ontology which they don’t even have any interest in a big proportion of it. Each user can only access parts of ontology she has interest in and is also allowed to.

Saving costly computation process
Since ontologies are located in central semantic data storage the costly computation process of labeling the consequences is happening only at server side and users are not demanded to be equipped with powerful computational facilities to be able to use Cloud-VPOS services. Also since the server side is hosted by powerful cloud infrastructure the computation process happening faster and response time to user requests decreases considerably.

Updating semantic models
Users are not concerned about the version of semantic models anymore. Whenever there need to be any change or update on ontologies these changes are happening at central semantic data storage. Since users
query the central server they immediately receive the latest version of semantic models. As a result ebbits components (users) wouldn’t be involved individually in maintaining and deploying of ontologies and it is happening only at server side which reduces the costs of maintenance.

**Role based access to ontologies**

In the new implementation accessing to ontologies is role based and each user is allowed to access the ontology according to her job role. Having users with limited access rights would eliminate security risks considerably because they won’t be able to access the whole ontology anymore and probably the parts which they are not authorized to.

### 5.2 Evaluation of the implementation

Implementation of Cloud-VPOS was accompanied with a case study about car manufactory. This case study would clearly demonstrate Cloud-VPOS services and role based access to ontologies. The goal of this case study was to provide a more obvious picture about functioning procedure of Cloud-VPOS as well as fulfilling the demands of ebbits project. Indeed car manufactory case study was a measurement for evaluation of implementation phase. In this case study users are accessing RDF recourses stated in the car manufactory ontology according to their job role and expertise.

Cloud-VPOS does not deal with axioms and OWL ontologies anymore. Instead it reads RDFS ontologies and return RDF recourses associated to user label. The case study as a proof shows how inference is done in the new implementation and how labels are calculated for new consequences. Each car technician can observe the portion of ontology associated to his label and also inferred consequences which are categorized to his label. During the review of case study the hierarchy of users and the amount of information they are allowed to observe also is taken into consideration.

#### 5.2.1 Inferred consequences

Here in the Figure 5.1 you can take a closer look at web interface of Cloud-VPOS. As you can see it is developed for the case study of car manufactory. Different types of technicians are able to choose their category of expertise (As can be seen in the Figure 5.2) to access resources from ontology associated to their context.
While user is selecting his area of expertise it’s also possible to choose about inference. This option allows user to decide if he wants to see only the resources stated in the ontology or he plan to observe inferred consequences in addition. If user chooses results including inference, Cloud-VPOS computes consequences and labels as well and returns triples of car manufacturer ontology relevant to his expertise and also consequences which have been labeled with his context. In the following figures the procedure will be demonstrated in more details.

Figure 5.3 displays the sub-ontology visible for Electrical Technician. To be more accurate and show the difference between results when inference is chosen, the output results would be filtered with “Airbag-
Sensor” which has already been demonstrated as part of ontology graph in the Figure 4.7. As it can be seen in the Figure 5.4 there are two assertions explicitly stated about Airbag-Sensor in the car manufactory ontology. But when user chooses to see results including inference he would be able to see not only assertions already available in the ontology but also inferred consequences regarding Airbag-Sensor labeled with his context authorizing him for observation. As it is shown in the Figure 5.5 the inferred consequences confirms the hierarchy of ontology graph presented in the Figure 4.7.

Figure 5.3: Sub-ontology for Electrical-Technician

<table>
<thead>
<tr>
<th>Sub-ontology for Electrical-Technician consists of following Axioms:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Coolant-Temperature-Sensor</strong> <a href="http://www.w3.org/2000/01/rdf-schema#subClassOf">http://www.w3.org/2000/01/rdf-schema#subClassOf</a> Sensors</td>
</tr>
<tr>
<td><strong>Miscellaneous-Auto-Parts</strong> <a href="http://www.w3.org/2000/01/rdf-schema#subClassOf">http://www.w3.org/2000/01/rdf-schema#subClassOf</a> car</td>
</tr>
<tr>
<td><strong>Clinometer</strong> <a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a> Clinometer</td>
</tr>
<tr>
<td><strong>Ignition-Switch</strong> <a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a> Ignition-Switch</td>
</tr>
<tr>
<td><strong>Body-And-Bayot</strong> <a href="http://www.w3.org/2000/01/rdf-schema#subClassOf">http://www.w3.org/2000/01/rdf-schema#subClassOf</a> car</td>
</tr>
<tr>
<td><strong>Gauges-and-Meters</strong> <a href="http://www.w3.org/2000/01/rdf-schema#subClassOf">http://www.w3.org/2000/01/rdf-schema#subClassOf</a> Electrical-and-Electronics</td>
</tr>
<tr>
<td><strong>Coolant-Temperature-Sensor</strong> <a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a> Coolant-Temperature-Sensor</td>
</tr>
<tr>
<td><strong>Airbag-Sensor</strong> <a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a> Airbag-Sensor</td>
</tr>
<tr>
<td><strong>Switches</strong> <a href="http://www.w3.org/2000/01/rdf-schema#subClassOf">http://www.w3.org/2000/01/rdf-schema#subClassOf</a> Electrical-and-Electronics</td>
</tr>
<tr>
<td><strong>Engine-Sensor</strong> <a href="http://www.w3.org/2000/01/rdf-schema#subClassOf">http://www.w3.org/2000/01/rdf-schema#subClassOf</a> Sensors</td>
</tr>
<tr>
<td><strong>Distributor</strong> <a href="http://www.w3.org/2000/01/rdf-schema#subClassOf">http://www.w3.org/2000/01/rdf-schema#subClassOf</a> Ignition-System</td>
</tr>
<tr>
<td><strong>Electrical-and-Electronics</strong> <a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a> Electrical-and-Electronics</td>
</tr>
<tr>
<td><strong>Powertrain-and-Chassis</strong> <a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a> Powertrain-and-Chassis</td>
</tr>
<tr>
<td><strong>Audio-Video-Devices</strong> <a href="http://www.w3.org/2000/01/rdf-schema#subClassOf">http://www.w3.org/2000/01/rdf-schema#subClassOf</a> Electrical-and-Electronics</td>
</tr>
<tr>
<td><strong>Door-Switch</strong> <a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a> Door-Switch</td>
</tr>
<tr>
<td><strong>Fuel-Gauge</strong> <a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a> Fuel-Gauge</td>
</tr>
<tr>
<td><strong>Electrical-and-Electronics</strong> <a href="http://www.w3.org/2000/01/rdf-schema#subClassOf">http://www.w3.org/2000/01/rdf-schema#subClassOf</a> car</td>
</tr>
<tr>
<td><strong>Ignition-Switch</strong> <a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a> Ignition-Switch</td>
</tr>
<tr>
<td><strong>Ammeter</strong> <a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a> Ammeter</td>
</tr>
<tr>
<td><strong>Distributor</strong> <a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a> Distributor</td>
</tr>
<tr>
<td><strong>Sensors</strong> <a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a> Sensors</td>
</tr>
</tbody>
</table>
Semantic Web mechanisms in Cloud Environment

Saeed Haddadi Makhsous

5 Results

2014-07-22

5.2.2 Hierarchy of users

Based on algorithm Cloud-VPOS has been implemented, each user context not only has access to sub-ontology relevant to that context label but also is able to observe sub-ontologies associated to user contexts with labels higher or equal to its label. The hierarchy of user labels has been specified on the lattice file produced for the car manufactory case study. Figure 4.8 demonstrates the car manufactory lattice for user roles and the hierarchy of their association. Based on the car manufactory lattice users with Body-Technician label are allowed to observe Auto-technician sub-ontology rather than theirs.

To verify the access rights of different users here an example is provided to show how a Body-Technician is also able to observe sub-ontology associated to an Auto-Technician. For this purpose the sub-ontologies of different users are summarized with triples which have “subClassOf” as their predicates, presenting simpler comparison. Figure 5.6 demonstrates sub-ontology for Auto-Technician including four triples while Figure 5.7 shows sub-ontology for Body-Technician. As can be seen the
Body-Technician not only can access triples associated to its context but also triples from Auto-Technician sub-ontology is visible for him.

Figure 5.6: Sub-ontology for Auto-Technician filtered for triples with “subClassOf” as predicate
Figure 5.7: Sub-ontology for Body-Technician filtered for triples with “subClassOf” as predicate
6 Conclusions
To conclude, the work which has been planned in this thesis project completed. The goals of the project has been yielded regarding re-implementing VPOS with new ontology API platform capable of using semantic data store, solving issues about scalability and usability/maintainability, and compromising semantic web services with cloud computing. Cloud-VPOS as final production of this thesis work has been developed to present context dependent views of ontologies as cloud services via Internet. Here more details about project achievements are presented.

6.1 Re-Implementing of VPOS with new framework
The main and initial step in this thesis project was to re-implement the VPOS with a new framework equipped with possibility of connecting to semantic data store rather than using file-system as semantic storage. Since main task of VPOS is to deal with semantic models and ontologies it was necessary to use a new ontology API not only capable of interacting comprehensively with ontologies but also offering chance of retrieving ontologies from semantic data store. Apache Jena framework was the best candidate for re-implementation since it meets all requirements and necessities.

VPOS was re-implemented with Jena framework and as a result it was not limited to only deal with axioms anymore. Jena framework empowers VPOS to consider triples as basic unit of semantic data, making the application much more compatible to other standards and frameworks. Meanwhile Jena was used to make a new version of VPOS capable of connecting to triple store.

6.2 Semantic data store as central data storage
The re-implementation of VPOS leads to using triple store as semantic data storage. Local file-system was replaced with remote central storage granting higher efficiency, security, and performance to Cloud-VPOS. All ontologies were moved to HANA triple store using Jena SDB (another component of Jena framework) to make them accessible via cloud
platform. In the new architecture triple store is actually a separate
database server which is hosting triples of RDFS ontologies.

6.3 Correcting issues about scalability and
usability/maintainability
Cloud-VPOS has the capacity of functioning over ontologies with any
possible size. The association to local machine memory is no longer a
matter of fact. Ontologies are stored in HANA semantic data store and
Cloud-VPOS utilizes Jena application programming interface to create a
reference for accessing them. The central storage of ontologies simply
means that there is only one copy of them available at Cloud-VPOS
server which can be modified or updated at any time and users are
guaranteed to receive the latest version of ontologies with no more
efforts for store and maintenance.

6.4 Compromising semantic mechanisms with cloud
environment
Cloud-VPOS is considered as an application which is dealing with
semantic models and functioning over cloud infrastructure. Cloud-
VPOS were developed as a web application capable of running on
HANA Cloud PaaS (Platform as a Service) or any other cloud runtime
environment. The goal of the project was to compromise semantic
mechanisms with cloud environment which has been achieved by pre-
senting context dependent views of ontologies as cloud services. Each
user which is using Cloud-VPOS services is now able to access the sub-
ontology associated to her context through web and Internet.

6.5 Case study of Car Manufactory
While developing procedure of Cloud-VPOS was going on the case
study of car manufactory was generated to meet the requirements of
ebbits project (as mentioned earlier car manufactory is an area of focus
in ebbits project). To fulfill this demand the car manufactory RDFS
ontology was produced containing RDF resources about different parts
of car such as door, sensor, engine, cooler, and etc. This ontology was
used to show how users with different context (Job role in this case) can
observe the sub-ontology relevant to their context as views of car manu-
factory ontology.
6.6 **Web interface for Cloud-VPOS**

Deploying the project on the cloud leads to creating Cloud-VPOS as web application which certainly needs a web interface as well. The web interface was developed using JSP web technology and its purpose was to facilitate accessing services presented by Cloud-VPOS. The design of web page was according to job role categories stated in car manufactory ontology. The web page is offering the possibility of choosing among different car technicians in order to return relevant sub-ontology.

6.7 **Possibility of including inference to results**

In the implementation of VPOS the possibility of using inference was not practically developed and it was only summarized for some test units. However in the implementation of Cloud-VPOS inference was considered as key structure to be developed in such a way that users would be able to choose if they want to observe the sub-ontology containing inferred consequences. The feature was also added to the web interface of application enabling users to follow their preference.

6.8 **Ethics perspective**

Virtual Private ontology Server (VPOS) is middleware software of ebbits software platform developed by SAP AG as a member of ebbits Consortium. Cloud-VPOS as re-implementation of VPOS was developed as a research project within SAP AG and the only authority responsible for publishing and distributing of Cloud-VPOS is ebbits Consortium.

Since Cloud-VPOS is dealing with semantic models (ontologies) the privacy and confidentially of semantic data can be matter of importance. When ontologies are about medicine or city information the associated authorities like hospital or municipality should take more security measurements in order to guarantee the confidentiality of patients’ information or city municipality plans. In very special cases like medicine ontologies taking privilege from patients might be taken into consideration since these ontologies containing private information only authorized to be observed by patient, doctor and associated authorities.
Thus, before applying the application into practice some certain security
measurements should be taken into consideration to guarantee the
confidentiality of data about people or authorities which are involved in
the ontologies utilized by Cloud-VPOS.

6.9 Future work

There are many possibilities for improvement in Cloud-VPOS which can
be taken into consideration. Here are some suggestions as future works
resulting into better performance and more comprehensive software:

- Replacing DBMS-backed data store with native triple store which
  rely on native database engine as persistent storage system, op-
timized for RDF processing.

- Equipping Cloud-VPOS with a dynamic web interface. Now the
  web interface of Cloud-VPOS is hard coded as static page for car
  manufactory case study. It can be upgraded in such a way to pre-
  sent the contents of the page dynamically based on the semantic
  model.

- Cloud-VPOS is designed for RDFS ontologies while as a further
  progress OWL ontologies also can be included enabling Cloud-
  VPOS to offer context dependent view on all sort of ontologies.
References


References


Semantic Web mechanisms in Cloud Environment
Saeed Haddadi Makhsous

References

2014-07-22


