Context Aware Services

YOUNES OUKHAY

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2006

COS/CCS 2006-3

Context Aware Services

OUKHAY Younes

January 25th 2006

School of Information and Communication Technology (ICT)
Royal Institute of Technology (KTH)
Stockholm, Sweden

Supervisor : Gerald Q. Maguire Jr.
Examiner : Gerald Q. Maguire Jr.

Abstract

Today customization of services and applications is one of the major challenges in facilitating ease of use. More and more people are interested in context aware services. In this work, I will study context awareness: What contributions can it make? What technical issues are raised? I will concentrate on the semantic problem and show how new technologies such as a web ontology language can facilitate creating context aware services. An application was implemented using these principles as a proof of concept and to enable some evaluation of this approach. This application has shown that combining semantic processing with SIP call processing is feasible and measurements have demonstrated a highly scalable context aware application for at least simple CPL scripts.

Sammanfattning

I dagens läge är en av de stora utmaningarna att skräddarsy applikationer och tjänster efter kundens begär för att höja användarvänligheten. Fler och fler kunder intresserar sig för tjänster som är kontextuppmärksamma. Jag kommer i mitt arbete att studera detta, mina frågeställningar är huvudsakligen, hur är detta användbart? Vad slags tekniska problem uppstår kring applikationen av detta koncept? Jag kommer att koncentrera mig på semantiska problem och visa hur nya teknologier så som webbontologispråk kan underlätta då man skapar tjänster som är kontextuppmärksamma. Jag kommer även att skapa en applikation där jag använder dessa principer för att visa konceptet och för att underlätta för en utvärdering av detta tillvägagångssätt. Denna applikation samt mätdata visar att det är möjligt att kombinera semantisk bearbetning och SIP anropsbearbetning för att skapa oerhört robusta applikationer, åtminstonde för enkla CPL script, som är hållbara i ett stort omfång av systemstorlekar.

Table of contents

List of figures

1 Introduction

1.1 Context aware services

1.1.1 What is context?

Many definitions of "context" can be found in the literature. Most definitions are too specific and incomplete because they are too closely related to the application it was used in. Here we will use Dey and Abowd's [1] because it is one of the most generic.

> "Context is any information that can be used to characterize the situation of an entity, where an entity can be a person, place, physical or computational object". [1]

Usually, to process context with computers, we define several types of contexts. The following categories are those used by Göker and Myrhaug [2]:

Spatio-temporal context	Where is the user? (At work, at home…). What is the time of the day (8AM, lunch time…)? Location has been frequently been used in context aware applications as it is one of the most useful context information.
User preferences	Provide profile (i.e. what the user likes and dislikes, the languages the user speaks…)
Social context	The user's friends and relatives, the user's social position…
Environment context	Information having to do with user surroundings and environment in a very general sense. It could include the available networks, the ambient temperature…
Personal context	What the user is interested in (finance, sports…), I s the user diabetic, does the user have allergies?
Task context	What the user is doing or will be doing.

1.1.2 What is a context aware service?

A service that uses context information as an input and adjusts it behavior accordingly is a context aware service. We can give examples of such services:

- If we use a presence detector, the temperature of the room can be automatically regulated to save energy by avoiding heating and cooling if the room is unoccupied.

- The network connection a PDA is using can be automatically changed according to the user's preferences and location: for example, when a user is at work, he wants his PDA to switch to the Bluetooth network instead of the wireless LAN except if he is using VoIP

- Upon a conference starting, all but emergency incoming calls are redirected to the user's voice mail

Services could be invoked by the user or automatically proposed to him according to his profile and situation. To be able to provide context aware services, you first need to collect all the relevant context information you have. This information is generated by sensors. Here we define sensors in a very broad way. Generally a context sensor is any context information provider. It could be a presence detector or a program informing you about how much free space is left on your disk(s).

Context enables the personalization of applications, thus providing higher quality services. Context aware services are a significant step toward ubiquitous computing. Context use can reduce user interaction and ideally it would end up enabling a truly pervasive and invisible computing environment. According to Mark Weiser [3], ubiquitous computing is "a new way of thinking about computers, one that takes into account the human world and allows the computers themselves to vanish into the background".

Using context is a very attractive concept and would provide very powerful applications, however, to this point, it is seldom used. Context unleashes many possibilities; but in fact, is rather difficult to handle.

1.2 Technical issues

1.2.1 Collecting context

Sensor diversity is an issue that has to be addressed. The system needs to adapt to each sensor, as frequently each sensor has its own communication protocol. Another difficulty is the format of the data collected. Today, this format is very heterogeneous: the size and type of data, the updating frequency, the language used, the measurement units, can vary from a context item to another. We need to be able to describe information and provide it to applications in a more generic manner. Therefore, we want a flexible system that has the ability to define new context information in such a way that new applications can readily use this data.

Additionally, there is a wide diversity of devices that can be used to access these services, i.e., some with more or less memory, computation power and bandwidths. Therefore we need to take these differences into account.

1.2.2 Architecture

One of the greatest challenges to context aware services is how to define an architecture that supports such applications. What kind of architecture is best suited? Should it be centralized, distributed, or both? Should the network have some intelligence or only the end user devices?

We can find in the literature 3 types of approaches to build a context aware system:

- Direct access to sensors
- Use of middleware
- Use of a context server

The first approach is pretty straightforward, as applications directly access the sensor data. Each application needs to use the correct driver to communicate with each specific sensor. The application must estimate the appropriate data "freshness" in order to know when to query the sensor. This approach has several disadvantages:

1. It is not a modular architecture hence it is hard to maintain or to add new features.
2. Each application that processes data must also gather it.

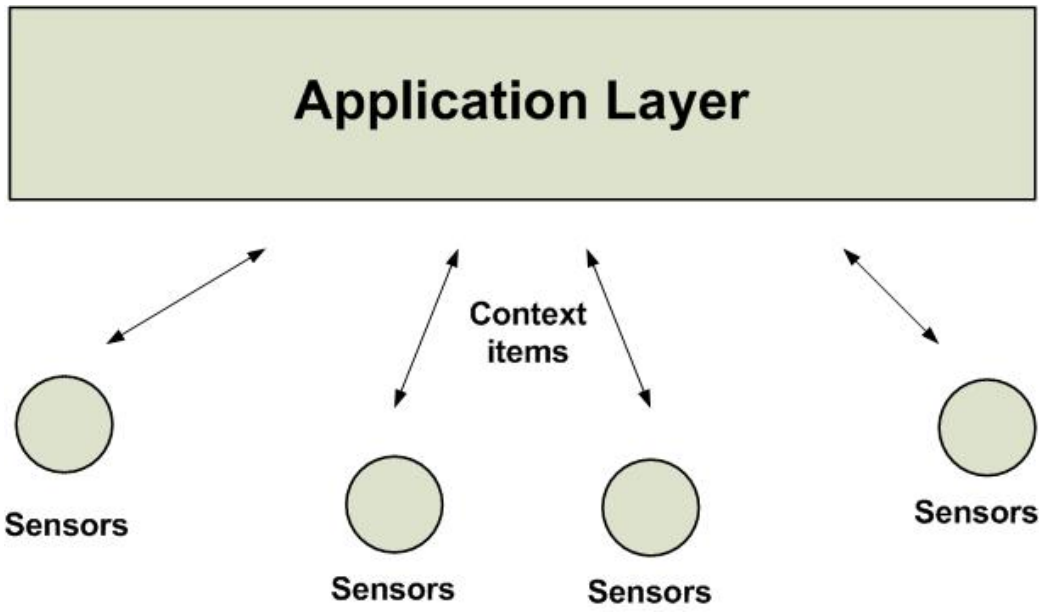

Figure 1.1 : Direct access approach

The second approach seems to be more efficient, although it requires a context infrastructure. The approach inserts an abstraction layer between applications and sensors. This enables applications to focus on their role; i.e. providing a context aware service. The middleware is now in charge of collecting the data, transporting it, updating it, and guarantee its security. Middleware has the advantage of being very generic, but we have to keep scalability issues in mind during the whole design process. While middleware is an elegant solution, it makes the system much more complicated.

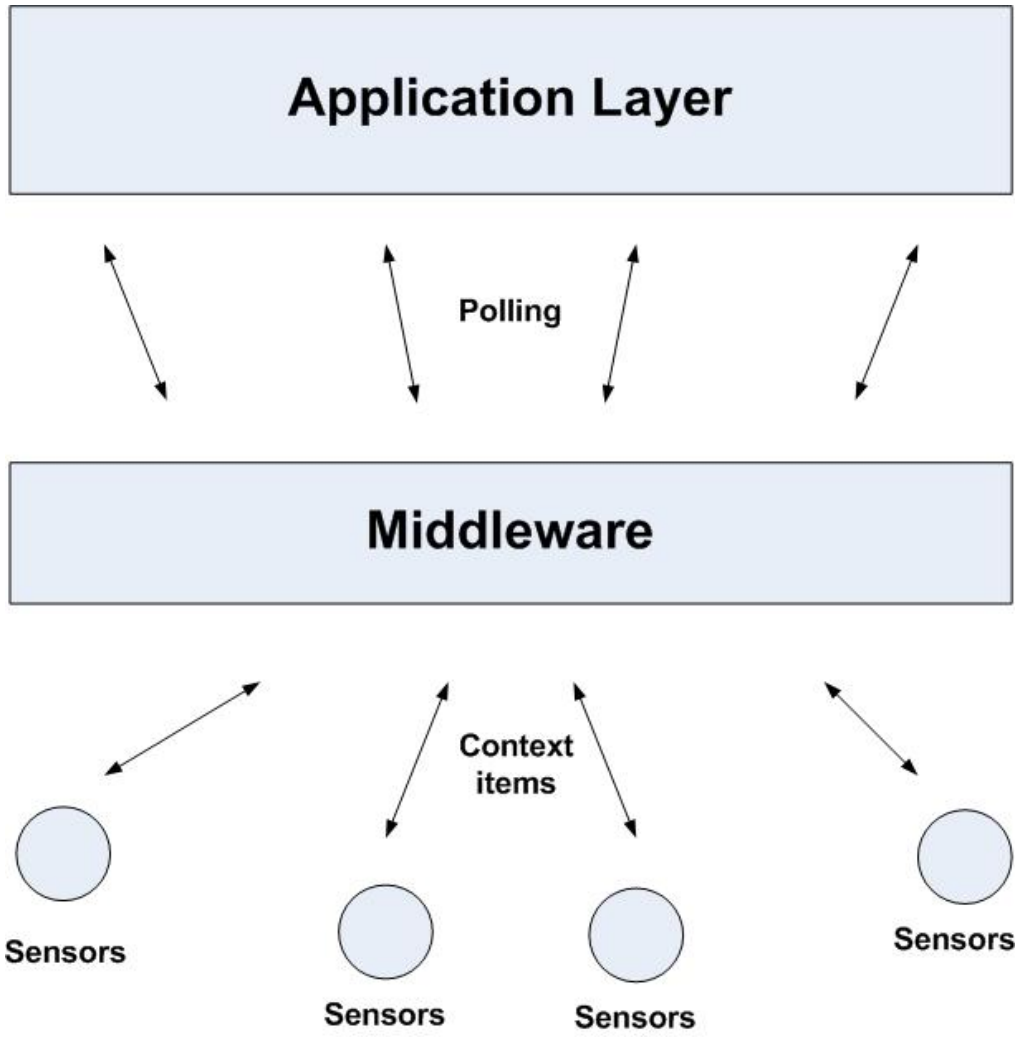

Figure 1.2 : Middleware approach

The third solution, i.e. the use of a context server, provides an intermediate solution between direct access and indirect access. In this approach the context server is responsible for forwarding messages from the sensors to the applications. Applications only need to poll the server in order to collect context data and to be context aware. The result is much less demanding than direct access, but lacks the generality of middleware. The cost is a different type of server for each type of context information.

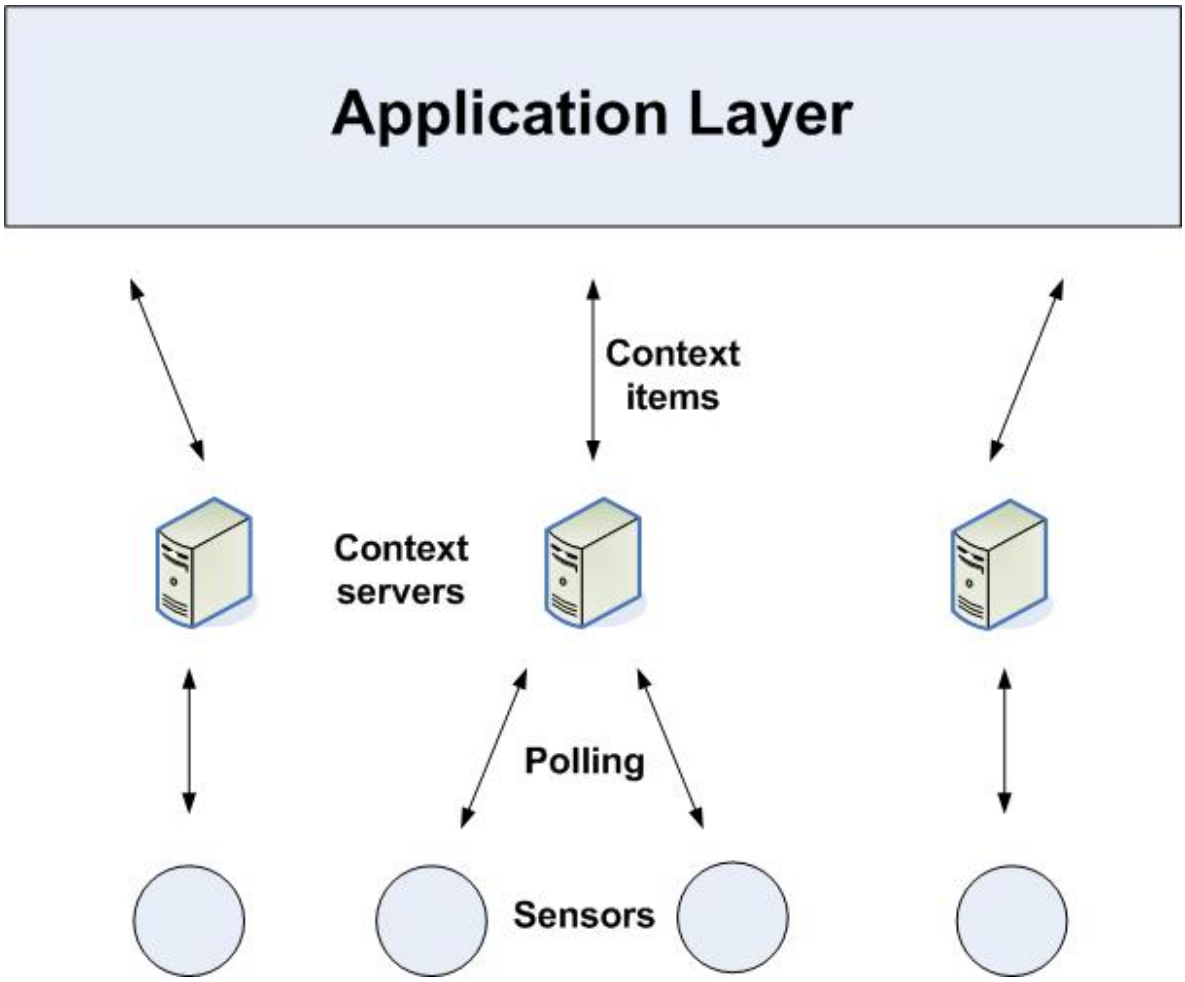

Figure 1.3 : Context server approach

These three approaches are not mutually exclusive. We often find a combination in the solutions we encounter.

1.2.3 Data description format

We have seen that information could have different formats (a noise level, a state: being on meeting, a preference: I do not smoke). The challenge is to describe all this information in a unified manner. Furthermore, this description should enable data transportation and permit data processing. Early user context applications used proprietary formats, then XML. XML is a step forward, but its expressiveness is still not great enough.

1.2.4 Context and time

Another challenge is context information updating. In some cases, context varies very rapidly and not always at a constant rate. Therefore, it is not sufficient to define a polling frequency and stick to it; as each context item has its own variance. For example, we rarely need to ask for the room temperature every minute. We also need a system that can detect and correct inconsistent information. Finally, we must not forget scalability.

1.2.5 Context information distribution

Context information distribution is one of the most difficult issues to address. Context information is initially in sensors, in the system, or in a device. Where are this context items exactly? Which context is relevant, for which application, how should we gather these items?

1.2.6 Refining context

Information provided by sensors is raw data. Often, it can be used as is by applications, other times it needs to undergo a refinement process. In that process, two or more items might be merged or a new value computed from multiple inputs. This process is very specific to the type of information and there is no unique refinement process.

1.2.7 Privacy

Privacy is very seldom addressed in implementations. However, we need to keep in mind that dealing with data such as the location of a person, his or her preferences, etc. could have legal considerations and may also make people reluctant to use the service.

Unfortunately, the less information we have about the user and their context, the less adaptation of service we can propose. Thus we must find a balance between what is needed and what is available.

Granularity as proposed in Harry Chen's work [4] could be an interesting approach:
The idea is to associate granularity parameters to context information items. For example, Alice does not want some people to know the specific room that she is in, but she is willing to let others to know the general description of her whereabouts. She might want her students to know she is on campus, but not which building she is currently in.

Often, the approach to privacy is wrong as developers first create their application, then try to add security and only consider privacy too late. Conversely, we think that security should be a primary concern from the conception phase of the context aware system. It is often quite complicated to deal with privacy and security, but it is clear that they require attention from the beginning. It is the condition sine qua none for a user to use the system and hence essential for the system to be widely adopted and to be successful.

1.2.8 Diversity of devices

Users connect to the Internet in different ways, via a PC, a laptop, a PDA, or a mobile. We have to consider this diversity and take into account the different storage capacities, computation capacities, additionally, physically small devices may present graphical and user interaction issues to solve. The solutions should be independent from any platform or operating system. The Jini technology [5] for example requires devices to have a java virtual machine which is not always possible and thus is not enough generic for context aware services needs.

1.2.9 Mobility and Wireless connectivity

Mobility management requires that we address the fact that users frequently move or are moving, resulting in loss of network connectivity.

1.3 Proposed solutions

1.3.1 Privacy

A very interesting solution to privacy problems has been proposed by the "ACAS project" [6] by use of an anonymizer proxy. This intermediary is positioned between the user and a service. It is a trusted third party who relays the dialog. This anonymizer creates new possibilities for many new applications as these applications do not need to be modified to provide anonymity.

1.3.2 Frameworks

Most of the context aware applications that have been created are too specific in terms of data description. This did not allow for the use of new context elements or adding a new service using the existing context elements. These applications are quite static and cannot evolve. One approach to this issue is to create first a framework for building context aware applications. The framework specifies how to get and handle context information, in terms of storage, acquisition, definition, and discovery and provides a complete design process for building such applications. These frameworks can be enriched with other solutions, for instance, they take advantage of the use of agents technology. Khedr Mohamed [7] provides a reconfigurable, interoperable, and adaptable agent-based framework that support context aware applications and the spontaneous situations occurring from explicit and implicit user's interactions with the environment.

Another advantage of frameworks is that they can handle uncertainty about the data collected from sensors. These frameworks are said to be probabilistic. They model uncertainty in order to prevent and correct its consequences. Data collected from sensors has a precision, a granularity and accuracy [8]. Precision represents the variation in the set of measurements sensors provide. Granularity represents the smallest unit that can be measured, and accuracy is the difference between the calculated value and the actual real value. Probabilistic frameworks' contribution is an assessment of the level of confidence the system can have in the collected data. MUSE [9] is an example of such probabilistic frameworks that uses a Bayesian modeling.

1.3.3 Scalability and performance

In order to achieve scalability, we have to take advantage of information caching. As it is very likely that a user goes to the same places, (home, work, shops…). By storing relevant data, we can significantly increase performance. For instance, the user's device could remember the sensors available at work, thus avoiding the need to (re-)discover them everyday.

Similarly, if we keep track of the recently invoked services and other personal information through caching, then history is accumulated. However, we need to have suitable mechanisms to ensure that this data is secure and relevant.

1.3.4 Middleware

Middleware is the best way to uncouple sensors and applications. Ideally, it would enable applications to collect context information irrespective of its type and what sensor it comes from, in a unified way. The OWL language can help in this matter as it provides a powerful data description format. In our implementation, OWL is used as an intermediary between context sources and applications. OWL and middleware can be used together. The middleware defines the architecture, the role and the interactions of the different components. It does not impose any data format.

Middleware can as well provide context information refinement or aggregation. In a more general manner, it enables to relieve the end users from the burden of context aware computation. In the proof of concept we are implementing, we are not going to use a middleware but rather use a more direct approach to get context information. We focus on how ontologies can make context aware services more efficient.

1.3.5 Context Servers

A context server collects context information from predefined sensors. It acts as a proxy to the applications that need context items. It is easier to set up and to manage than middleware architecture. The context toolkit [1] proposed a framework for building and rapid prototyping of application with such architectures. It makes an effort in uncoupling applications and sensors through the use of widgets.

1.3.6 Agents

Agents can contribute to context aware services. They can provide matchmaking between user requirements and the available services. They are also interesting because they can share information with other agents, which has two advantages:

1) Avoiding having a single node where all context information would be stored: as this node would become a single point of failure.
2) An agent need not acquire all the context information itself; other agents can help. Of course, this increases the complexity of the system, as the knowledge is then distributed.

Agents can learn about the user over time. They can also reason about context, and thus infer new information. This is very interesting as it enables us to determine and refine the choice of the best suited service.

Reasoning about context information can be used to create a knowledge database. Fortunately, the progress made in the agent domain in recent years is tremendous. An example of a reasoning using OWL is provided in the section 2.1.4.

An example of agent-based middleware is “An agent-based middleware for context-aware ubiquitous services” project [11] (AMUSE). In AMUSE, Agents cooperate to provide context aware services and pay particular attention to the quality of service (QoS) issue. An agent-based solution has been chosen here because it provides an effective way of exchanging information about the network connectivity and the quality of service.

1.3.7 Evolution of context aware services

Tackling the context awareness question is difficult. Context aware applications make use of many unsolved engineering and theoretical issues. The first applications realized were closed applications in terms of data structures used and were also too specialized: Adding a service or modifying one could be difficult. Then appeared the first middleware architectures and the use of XML for data structuring. Middleware is a step ahead for context aware services yet it does not bring an answer for all difficulties encountered especially interoperability. Eventually “service oriented middleware” and use of semantic languages came in. Context information can now be described in a powerful way (see section 3) and reasoning can be executed on context information especially through the use of agents.

There is still some way to go and context aware services are heavily dependent upon on the progress in the following fields:

- Sensors: need to longer operate and be both more specialized (for some tasks) and less specialized (for other tasks – thus having larger manufacturing volumes), and able to cooperate (generally as some sort of wireless sensor network)
- Middleware and distributed systems: to provide better context discovery, transport, and processing, and greater reliability
- Human computer interface: to exploit context sensing
- Semantics: context modeling and intelligent agents, for greater interoperability and reasoning

2 Background

2.1 Session Initiation Protocol

Session Initiation Protocol (SIP) [12] is an application layer protocol. It is an approved standard of the (Internet Engineering Task Force) IETF since 1999 and today the core protocol specification is documented in RFC 3261. As one can infer from its acronym, SIP is used for session initiation, but also modifying and ending sessions with one or more participants. It can as well notify applications of the presence of a user or be used to send short messages. SIP uses peer to peer communications, i.e. all participants are on the same level; there is no master or slave: Depending on the messages being exchanged, a participant can act as a client or as a server.

SIP was meant only for communication establishment. It does not provide any mechanism that realizes the communication itself. For such a purpose, we may use RTP if we want to transfer voice for example. Thus, SIP works in conjunction with other protocols: The Session Description Protocol (SDP) is used to describe and negotiate session information such as the encoding used, the protocol port, the multicast addresses etc

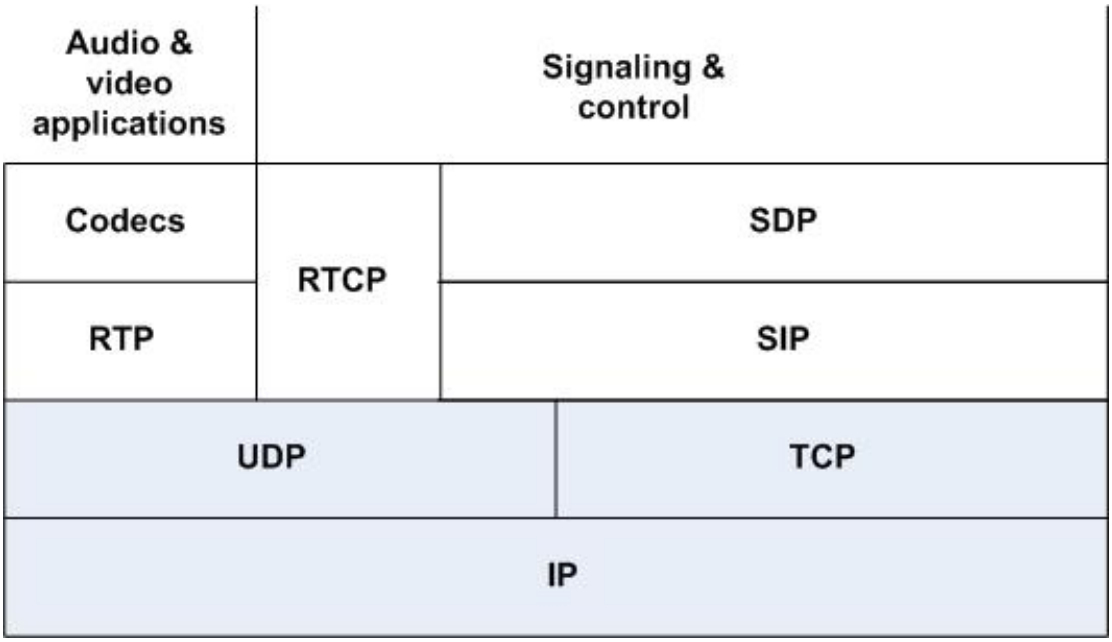

Figure 2.1 : SIP Protocol stack

SIP was designed to have an easy implementation, a great scalability, and to be flexible. SIP is a text-encoded protocol that is very similar to HTTP and SMTP regarding its syntax and transaction model. It contrasts with the complexity of H.323 signaling protocol, its counterpart in the telecommunication world. We can mention here that SIP has a different approach compared to the traditional PSTN. SIP moves the processing and logic to the end points.

SIP users are identified by their SIP URI (Uniform Resource Identifier). A SIP URI has the following form: sip:user@myhost.com, for example, sip:bob@iptel.org. Therefore, a SIP URI consists of a username and a domain name separated by the @ character. They follow the same pattern as e-mail addresses. It is even possible to use one's email address as a sip identifier making it easy to remember.

Let's now consider SIP architecture and describe all its components.

User Agents (UA) UAs are the end devices of the SIP architecture. User agents can be SIP phones or soft-phones (software emulating a SIP phone) running on a computer or mobile device. A UA is composed of a user agent client (UAC) and a user agent server (UAS). The UAC originates the requests and the UAS generates responses to the received requests.

SIP proxy servers proxy servers have many functions, they can perform routing of UA requests, they can implement authentication mechanisms, they can perform accounting etc

Redirect servers redirect servers receive requests from UA and proxy servers, then look up a location database and return responses indicating where request initiators should retry sending their requests.

Registrar servers registrar servers receive registration requests specifying a user new status (locations), and update accordingly the location database.

In practice, proxy servers, redirect servers, and registrar servers are merged into one entity. An example of a SIP dialog involving all the above components of the network is provided in section 4.6.1. For available implementations, please refer to [23] that maintains a list of them.

2.2 Extensible Markup Language

The Extensible Markup Language (XML) is a general purpose markup language. It is a World Wide Web Consortium recommendation. XML is a text format derived from SGML. It facilitates data sharing between different platforms but can also serve as a base for creating specialized markup languages such as RDF, OWL, and MathML etc

XML describes data in a hierarchical way, using tags to structure information. Elements are data containers that are defined by their name; they may also have attributes and a content.

Unlike HTML, XML has a strict syntax. We say an XML document is well formed if its syntax respects the XML specification constraints. For instance, any opened tag should be closed.

If a document is well-formed and complies with a particular schema (DTD or XMLSchema), it is said to be valid. Validity is hence a stronger constraint. A schema specifies a number of constraints on the structure and content that go beyond the basic constraints imposed by XML itself. We use software called "parsers" to determine if a document is well formed and valid.

2.3 Resource Description Format

RDF was proposed by the W3C's semantic web activity. As we said earlier, the Resource Description format (RDF) can be serialized into XML. But RDF is independent of XML. It is a metadata model that helps writing data descriptions. It uses a <subject><predicate><object> model, called "triple". The *subject* is what we describe, the *predicate* is a trait or aspect of the subject we are describing, and the *object* is the value of the trait. A triple would look like this if expressed in pseudo-syntax:

```
<Student X><is author of><master thesis Y>
```

RDF enables automated processing of information. Software can therefore store, exchange and classify data in an automated manner.

RDF has a semantic extension, the Resource Description Format Schema (RDFS) language. It enables checking an RDF resource's validity just like an XML Schema.

2.4 Web Ontology Language

OWL is defined by the W3C as a Web Ontology Language. It was designed for use by applications that need to process the content of information instead of just presenting information to humans. It can be used to describe the meaning of terms and the relationships between these terms. We call this representation of terms and their interrelationship an ontology.

OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, by providing additional vocabulary along with a formal semantics. We use OWL representation when we expect machines to perform useful reasoning tasks based on information or when we feel the need to go beyond the basic semantics provided by XML or RDF. OWL is hence situated at the semantic level.

OWL's expressiveness is greater than XML, and greater than RDF and RDF-S. In fact OWL is built on top of these languages as figure 2.2 shows. Thus, it goes beyond these languages in its capacity to represent machine interpretable content. RDF enables us to know that information exists and to know its properties. OWL lets you know if two items of information are the same, if its properties can have one or many values, and in a more general manner link two items of information to another. Here we will use OWL to describe data even though it is not web content.

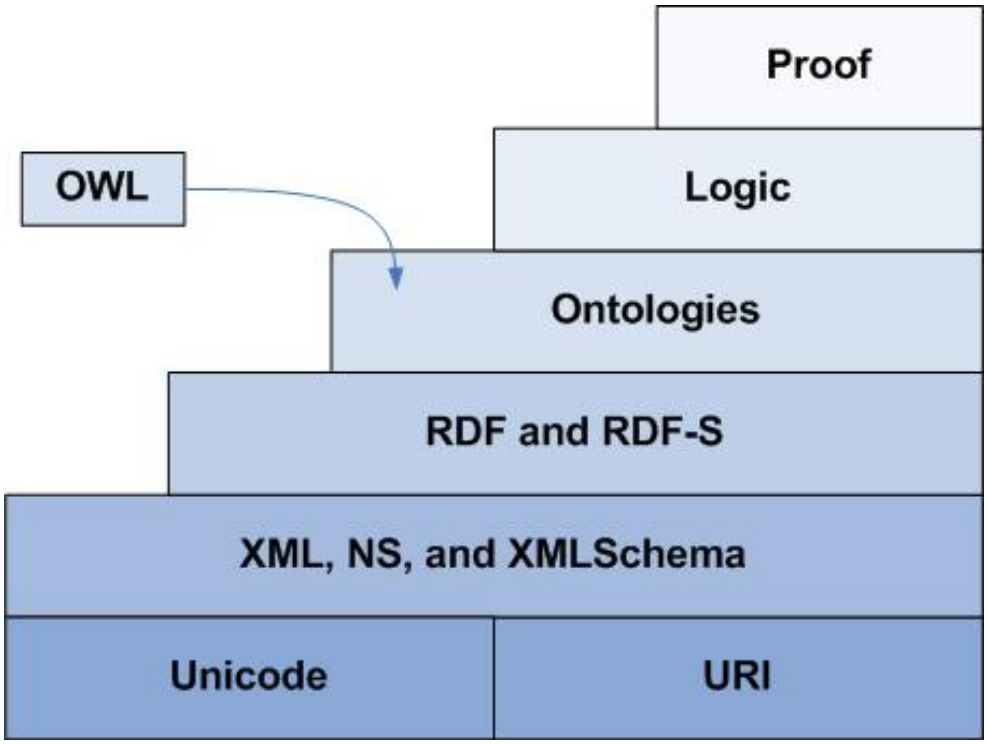

Figure 2.2 : Semantic layers

2.5 Call Processing Language

CPL [13] is a language that is used to write scripts for call handling. It is an IETF standard (RFC3880) and also an XML based language. Its greatest advantage is its simplicity. It was developed to allow non-trusted users to upload their service requests to be executed on SIP servers: It has no loops, no variables, and does not have the ability to run external programs. An alternative means of programmatically providing call handling is SIP Common Gateway Interface (SIPCGI). SIPCGI is more complex and is less secure as the language is not limited, hence arbitrary computations can be specified.

CPL also has a DTD that specifies the syntax it should respect. Therefore, the CPL scripts should be validated before they are uploaded to the server. Note that the CPL standard does not specify a means for script uploading. However, this upload can be realized in a secure manner, depending on the server. Here, the confidentiality of the request is required, but also it must be authenticated - so that only the user can redirect or otherwise process their call(s).

CPL enables the SIP server to provide many features, such as call blocking, call redirecting, and others -according to a number of predefined parameters which include time, caller, callee, etc. These functions can be applied to outgoing or to incoming calls. CPL can be used to specify quite complex features. For example, we can have a script which performs the following:

When a caller tries to reach user Y's desk phone, if Y does not answer within a certain amount of time, then calls Y's boss are forwarded to Y's mobile phone, and all other calls are directed to Y's voicemail.

The scripts used in this project are included in the appendix D.

3 OWL and context aware services

3.1 Using ontologies

We need to define a common ontology, in order to enable all systems to cooperate. Unfortunately, this is not easy -because we need everyone to agree upon a common specification. However, a great feature of OWL is that an ontology can point to another ontology and import it: all the classes, properties, and individual definitions that are specified in the imported ontology can be used in the importing ontology. This is what we have used in the implementation we realized: several OWL context files were used. Each user has his/her own OWL context file describing his/her context. To avoid defining for each file what a "user" is, what "context items" are, we import an ontology that defines these elements.
To do so, one can use the following OWL tag:

```
<owl:imports rdf:resource="http://172.16.1.6/owl/userContextOntology"/>
```

The attribute value corresponds to the namespace of the ontology. It can also specify its location. The context ontology we are using is described in section 4.3. Thus ontologies can be easily imported and shared. This is a step toward greater system interoperability. Several systems can have their own knowledge representation, but may still cooperate. An easy way to find existing ontologies on the web is to use the Swoogle search engine [24].

The general paradigm [10] for using ontologies in context aware services is to define an intermediary semantic layer between context sources and context users. As the sensors provide raw context items, we need some "adaptor" to format these context items into OWL context items that can directly feed the OWL context database. Similarly, we may define some context providers that can handle context information queries issued by context aware services. These context providers are in charge of querying the database and answering queries so as to provide services with high level context information. The context providers can take advantage of technologies such as OWL-S for service invocation for example (see section 3.3). We summarize this in the following figure:

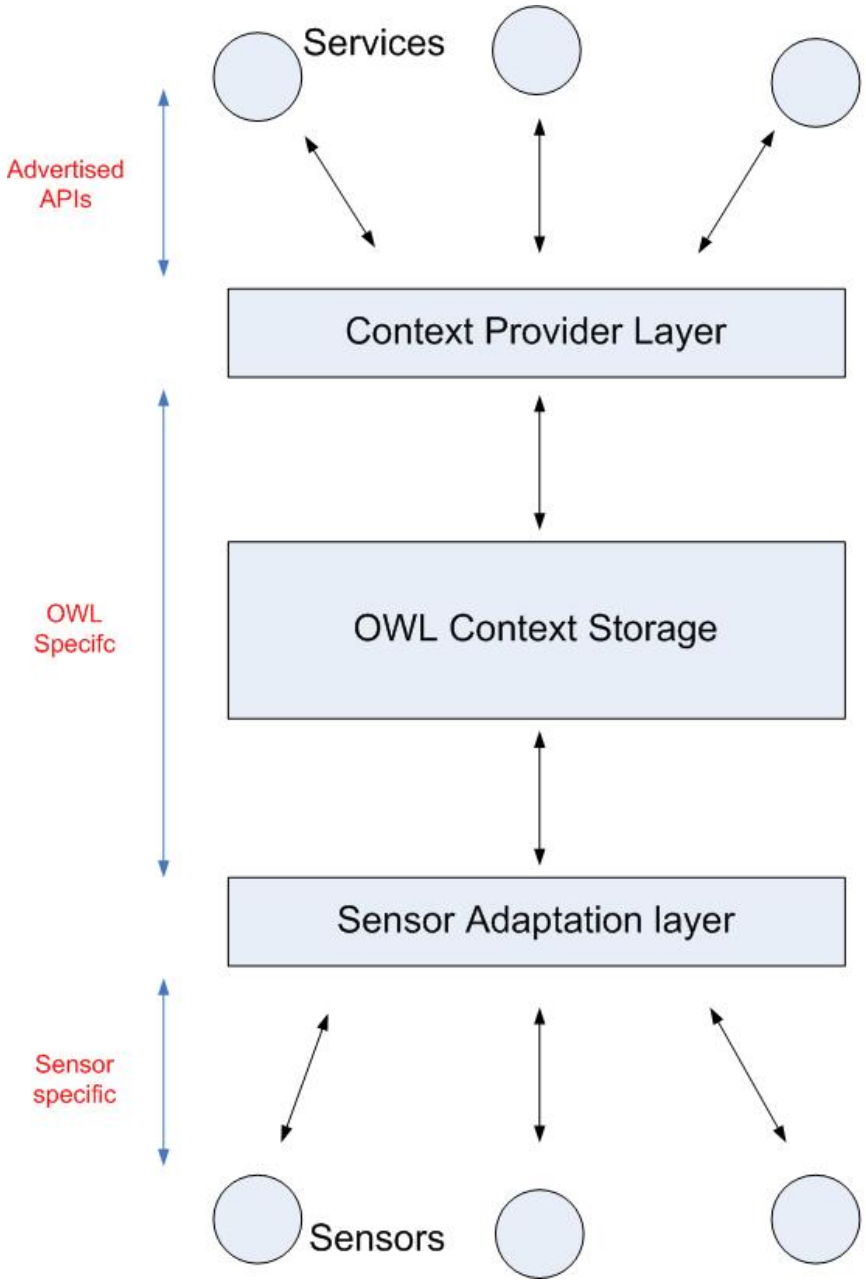

Figure 3.1 : The ontology architecture paradigm

Note that with these context providers, any service can query the context database, as it is an open architecture. Context providers can also implement access control mechanisms. This is necessary when users wish to protect their context data. One might not want others to know where you are and what you are doing. A very simple policy that does not require any management is: any user can access all context concerning himself and only context concerning himself. A second approach based on the first, would enable user A to define which context items he wishes to share with whom. For instance, the user Bob lets his boss (another user) know about his location (context item) during working hours.

This pattern can then be extended. In our implementation, we simply use an OWL file for context storage. The OWL file is then parsed to extract the relevant data we would like to use. This case is the most straightforward one, but we can subsequently take advantage of the use of a knowledge database to store our context information. Such a knowledge database has many interesting features. First, it makes reasoning on context much easier (see section 3.2).

Secondly it can prevent (or detect and correct) inconsistencies in the context information it manages. Inconsistencies are contradictions in the context information stored. For example, if a user's context indicates that he is showering and sitting in the living room, there may be inconsistency. Knowledge databases can be configured to tolerate more or less knowledge inconsistency and to keep the coherence of the gathered information [18].

3.2 Reasoning with OWL

We can distinguish two types of reasoning [15]. Reasoning based on ontologies, and reasoning based on context rules. Each of these approaches is described below.

3.2.1 Reasoning based on ontologies

Let's consider two examples:

1)

Provided:

- Grimeton is a meeting room
- Meeting rooms are working places

A machine can infer that:

→ Grimeton is a working place

2)

Provided:

- Jim belongs to the group "best friends"
- Best friends are also friends

A machine can infer that:

→ Jim is a friend

These two examples use "class inference" reasoning. Other types of reasoning based on ontologies can be found in [21].

These reasoning mechanisms use the OWL logic. They result from inferences contained in the ontology, i.e. in the definitions and interrelationships between the terms defined. Conversely, reasoning based on context makes use of predefined rules, which are external to the OWL ontology.

3.2.2 Reasoning based on context rules

In our implementation, we used only reasoning based on context rules as it was easier to implement. We have used a set of predefined rules to draw conclusions based upon the user's context. Here is an example:

Consider the user Bob; he has a context item that describes his location. This item is set to "Grimeton" and we have the predefined context based rule:

"If a user is in a working place, then he is busy, incoming calls from friends should redirected to Bob's assistant, and all other calls should be rejected"

Now, if Jim (friend of Bob) calls he will be redirected to the assistant because we can determine that Jim is a friend and that Bob is in a working place from the reasoning based on ontologies. Often, both types of reasoning are combined as in the above example. For details of the context rules we used, and examples of their use see section 4.3.

These second type of rules should be defined by the user. We can offer him a default configuration that he may modify and adapt to his needs.

3.3 OWL-S : OWL for services

OWL-S [17] is an ontology of services that enables users or their applications to discover, invoke, compose, and monitor Web resources offering particular services and having particular properties thus enabling a high degree of automation (if desired). OWL-S enables the description of the external behavior of a service via a semantic model, in which operations are described semantically in terms of inputs and outputs.

To put it in a nutshell, OWL-S markup of services provides:

- the information necessary for Web **service discovery** (could be specified as computer-interpretable semantic markup at the service Web sites) thus a service registry or ontology-enhanced search engine could be used to locate these services automatically.
- Automatic Web service invocation via **automatic invocation** of a Web service by a computer program or agent, given only a declarative description of that service, as opposed to when the agent has been pre-programmed to be able to call that particular service. Thus depending on the user's context, we can dynamically find a specific relevant service.
- We can provide automatic selection, composition, and interoperation of Web services to perform some complex task, given a high-level description of an objective

4 Case Study

4.1 Why build a prototype?

An application was developed to provide a service that will be context aware in order to prove that OWL can be used and that it facilitates designing context aware services.

4.2 The service chosen for implementation

The implementation of a service that is context aware using OWL was designed and implemented as a proof of concept. The example chosen is a context aware SIP proxy:

> We would like to modify the behavior of the SIP servers according to the context of the user. We need to acquire this context information from the OWL source (see section 4.4.4) that describes it, and then translate it into CPL code that is understandable by the SIP servers. Finally, we need to upload these scripts onto the servers.

4.3 Example Architecture

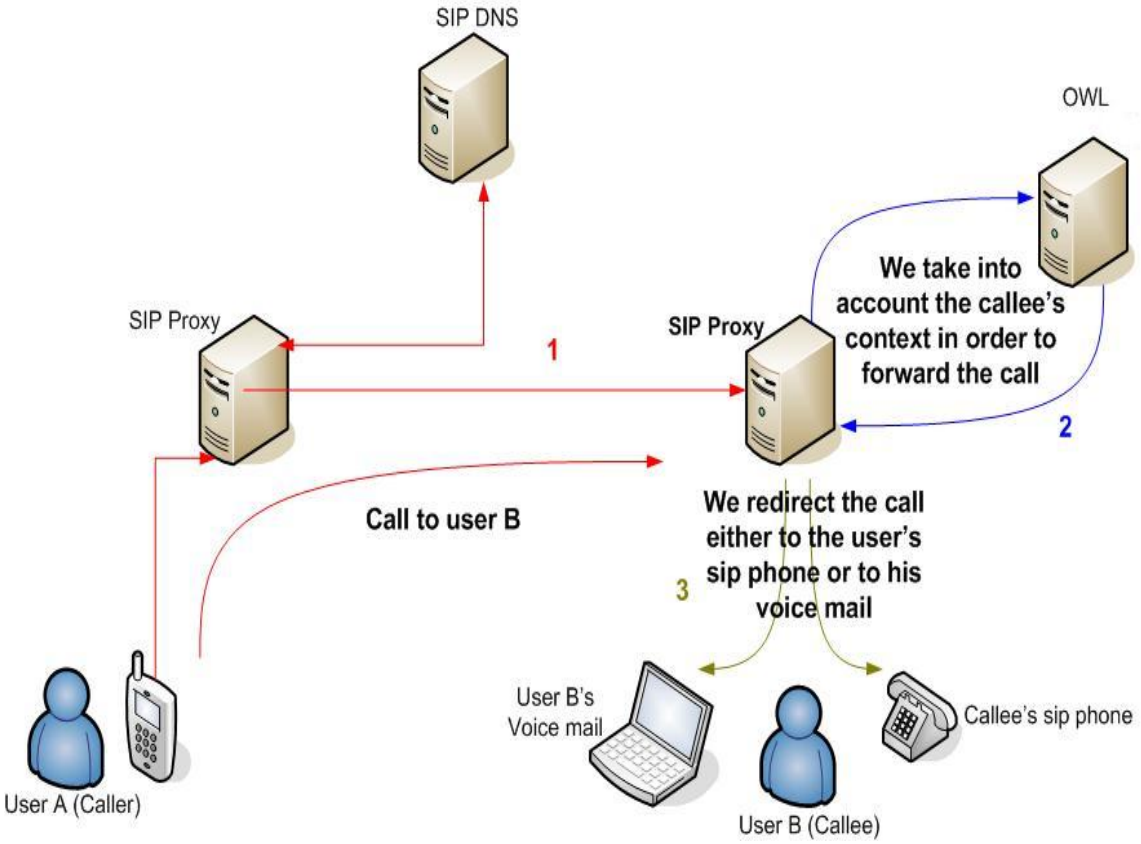

Figure 4.1 : Example architecture

The goal of this implementation is to use OWL as a context source. We do this because OWL provides an efficient way of describing and processing context information. The OWL context source is created and designed in a very generic manner. Therefore, it could later be a context source for other applications. It can be extended and even import other ontologies.

A context ontology has been designed for our service implementation. The context source is very simple; it consists of an OWL file that describes the context of a user according to this ontology specification (figure 4.2). The complete OWL file is in the appendix E.

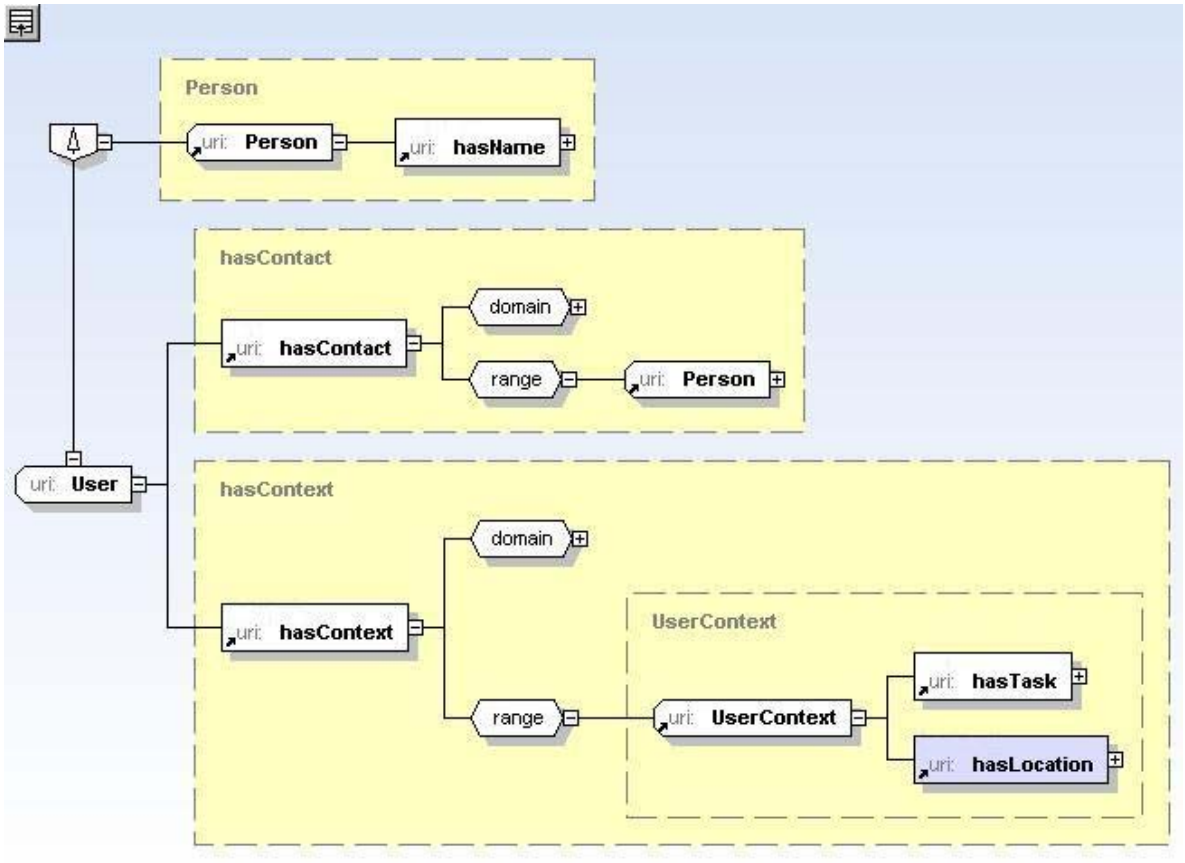

Figure 4.2 : The context ontology

We can interpret the above diagram as follows:

- A User (Class) has a name (as he inherits that property from the Person Class)

- A User has contacts; these contacts belong to the class Person.

- A User has a context (Context Class)

This Context class has two properties:

- property 1: hasLocation (place where the user is e.g. in Grimeton which is a conference room)

- property 2: hasTask (corresponds what the user is doing, e.g. on meeting)

We are using three following scenarii:

Scenario 1:
The user is in a meeting and does not want to be disturbed. All calls to him should be directly forwarded to his voicemail. For this scenario we use the CPL script called *callRedirectUnconditional*, (see Appendix D).

Scenario 2:
The user X is at work and does not want his friends Y and Z disturbing him. Therefore, only calls received from Y and Z will be redirected to the user's voicemail. While other calls will reach the user X. For this we use the script *screeningRedirectBasedOnCaller* (see Appendix D).

Scenario 3:
The user is now at home and does not want to receive calls from his colleagues who are at work (i.e., contacting him from the network domain mywork). All the incoming calls with a hostname that matches @mywork will be redirected to his voicemail. We use for the script *screeningRejectionBasedOnHost* (see Appendix D).

Depending on the context of the user, the application will upload one of these three scripts. The SIP proxy will then adapt its behavior.

This application does not provide a mechanism to collect context. Therefore, there should be another application in charge of collecting the users' context information, storing it and updating it. This storage should be in the form of an OWL file: the implemented application assumes that each user has his context file. First, it parses the OWL file to check its validity, and collects the context information items from it. Then it chooses according to this context data which script corresponds to the user's state. Eventually, it will upload this script on the server.

The figure 4.3 shows an example of an OWL file. The corresponding user is bob, and his property "hasTask" is set to "meeting".

```
<?xml version="1.0"?>
<rdf:RDF xml:base="http://www.mydefaultnamespace/younes"
xmlns="http://www.mydefaultnamespace/younes#"
xmlns:acas="http://www.mynamespace.com/owl#"
xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
        <owl:Ontology rdf:about="">
                <owl:imports rdf:resource="http://172.16.0.78/owl/userContextOntology"/>
        </owl:Ontology>
        <acas:User rdf:ID="bob">
                <acas:hasContext>
                        <acas:UserContext rdf:ID="bobContext">
                                <acas:hasLocation
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">grimeton</acas:hasLocation>
                                <acas:hasTask
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">meeting</acas:hasTask>
                        </acas:UserContext>
                </acas:hasContext>
                <acas:hasName
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">bob david</acas:hasName>
        </acas:User>
</rdf:RDF>
```

Figure 4.3 : OWL context instance

The correspondence between a set of context items (user, task, location) and a CPL script is a direct one. This application could be improved if it could generate scripts itself instead of choosing from a predefined set of scripts. OWL here will be very useful thanks to its reasoning capabilities.

Another improvement would be providing an API to the application that gathers the context items just like we have seen in section 3.1. That way, adding context data into the OWL database does not require from an application to know how we store our data whereas now, it needs to set the context item into the correct tag. This way, our application can also know when an update of context information has been realized and upload the corresponding script right after.

What we envision is to have an OWL based context source for all services: we have developed one service which is call processing but the goal indeed is to enable any kind of service to use this OWL based context database. Each user would have his own context source that contains his personal context. And all type of applications would query this database in order to be context aware and better suit the user's needs. Each application can either use the existing ontology or extend it if required.

A possible implementation is having this context database being a server that you host at home [22]. This way, many difficult issues related to privacy would be more easily solved.

5 Implementation and deployment

5.1 Using SER

5.1.1 Call setup example

Consider the four following machines:

- a machine hosting the caller UA (jim@xxx.xxx.xxx.248
- a machine hosting the caller SIP Proxy (xxx.xxx.xxx.247:5060 the SER server)
- a machine hosting the callee SIP Proxy (195.37.77.99, iptel.org proxy)
- a machine hosting the callee UA (jadesip@xxx.xxx.xxx.216)

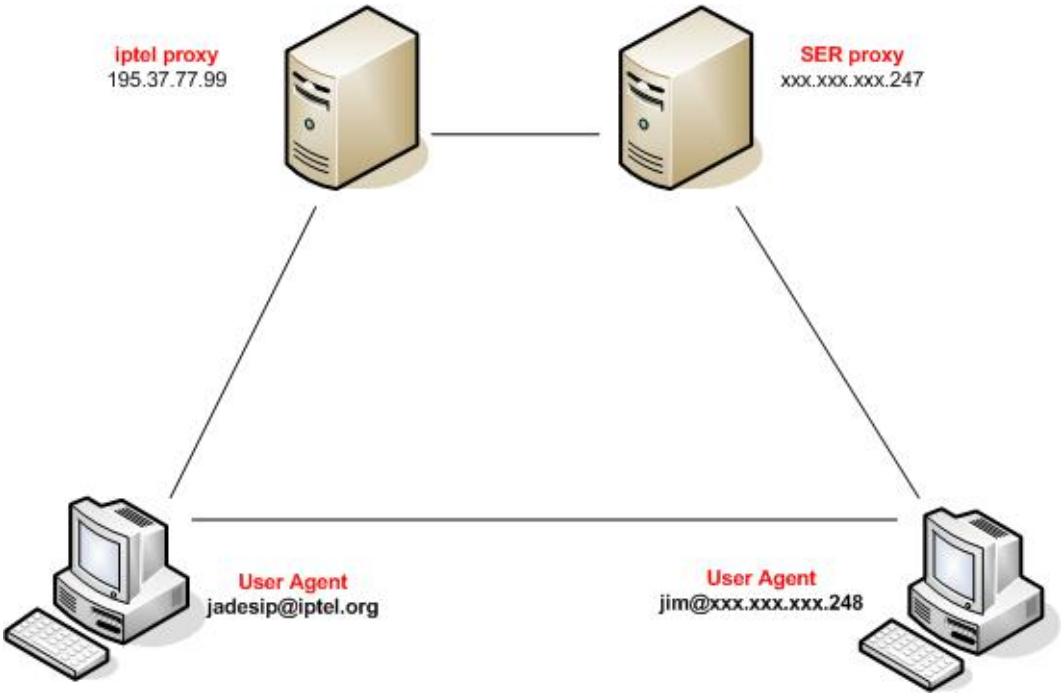

Figure 5.4 : Network Architecture

For the above configuration the callee needs, you need to register to have a SIP account at iptel.org. This registration is free of charge.

In this scenario, we will not be able to process jadesip's incoming calls as the iptel server does not belong to us and thus we cannot configure it. We have however used this example as it a comprehensive SIP session establishment with a DNS lookup.In the next configurations, we will later only consider scenarios where the call receiver is registered in our SIP (SER) proxy.

Figure 5.5 is obtained from ethereal and shows the SIP messages exchanged during a call from jim@xxx.xxx.xxx.248 to jadesip@iptel.org. Ethereal is available on Windows and Linux platforms. It is very useful to monitor network traffic and can be configured to monitor specific network interfaces.

Figure 5.5 : SIP dialog

In figure 5.6, we can see the different steps in the call setup:

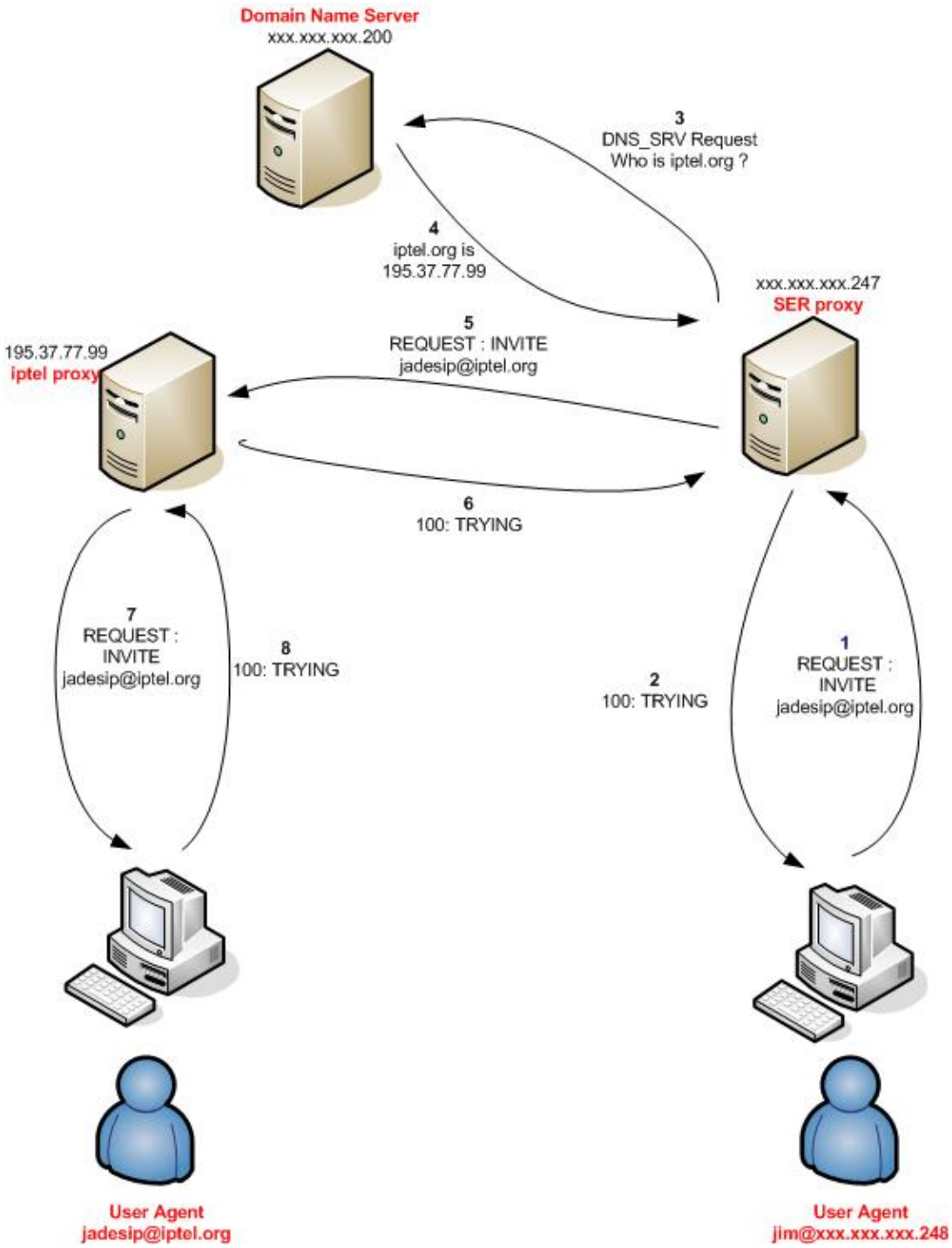

Figure 5.5 : Step 1, ringing the callee

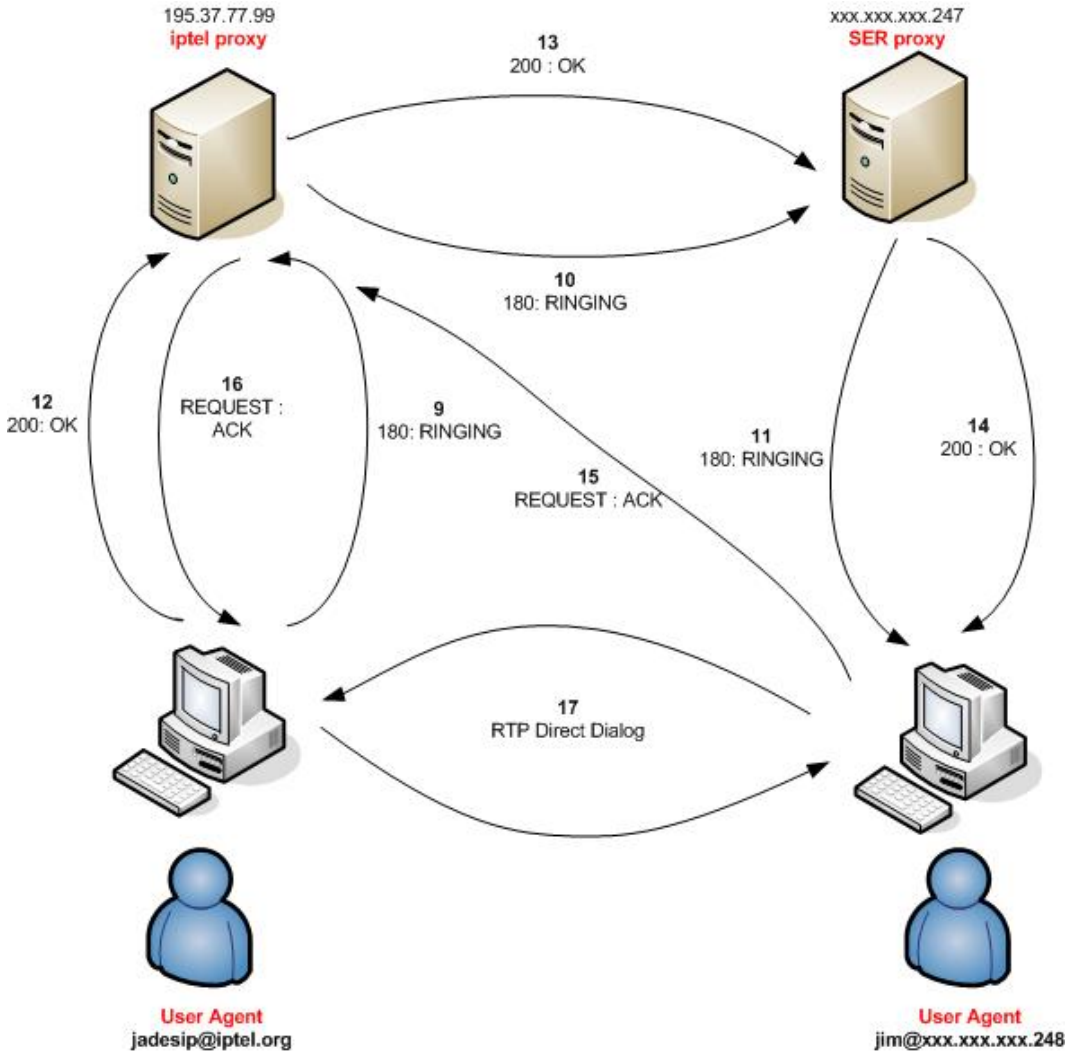

Figure 5.6 : Step 2, communication establishment

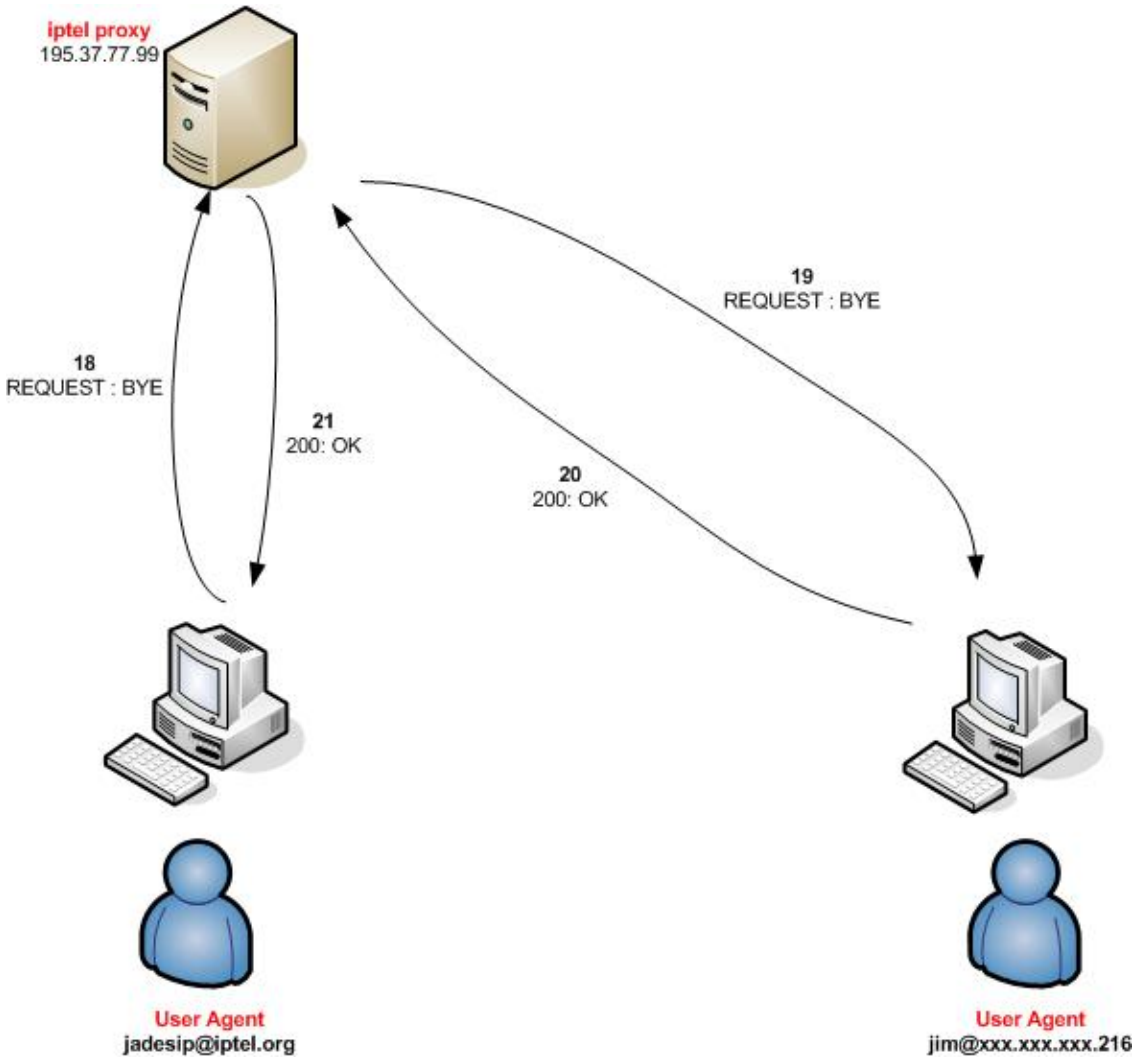

Figure 5.7 : Step 3, session ending

This is an example of a SIP session establishment. There are many other scenarii. In the next section, we will show how an incoming call is processed using CPL.

5.2 SER and CPL

In this second scenario, we will use a CPL script loaded by Bob to redirect a call. Jim is calling Bob, but Bob is in a meeting and all calls should be redirected to his assistant.

Figure 5.8 is obtained from ethereal and shows the SIP messages exchanged during this call. It is an unconditional redirection and the two users are registered to the same proxy.

Consider the two following machines and the cisco1 hardware UA:

- a machine hosting the caller UA (jim@xxx.xxx.xxx.202)
- a machine hosting the call SIP Proxy (xxx.xxx.xxx.247:5060 the SER server)
- a Cisco1 hardware UA (@xxx.xxx.xxx.222)

In this scenario, Jim tries to call Bob. As Bob is working, Jim's call is redirected to the hardware SIP phone cisco1.

We can see in figure 5.9 that redirection takes place when the server send the 301 "MOVED PERMANENTLY" message. The rest of the dialog is a classical SIP session establishment.

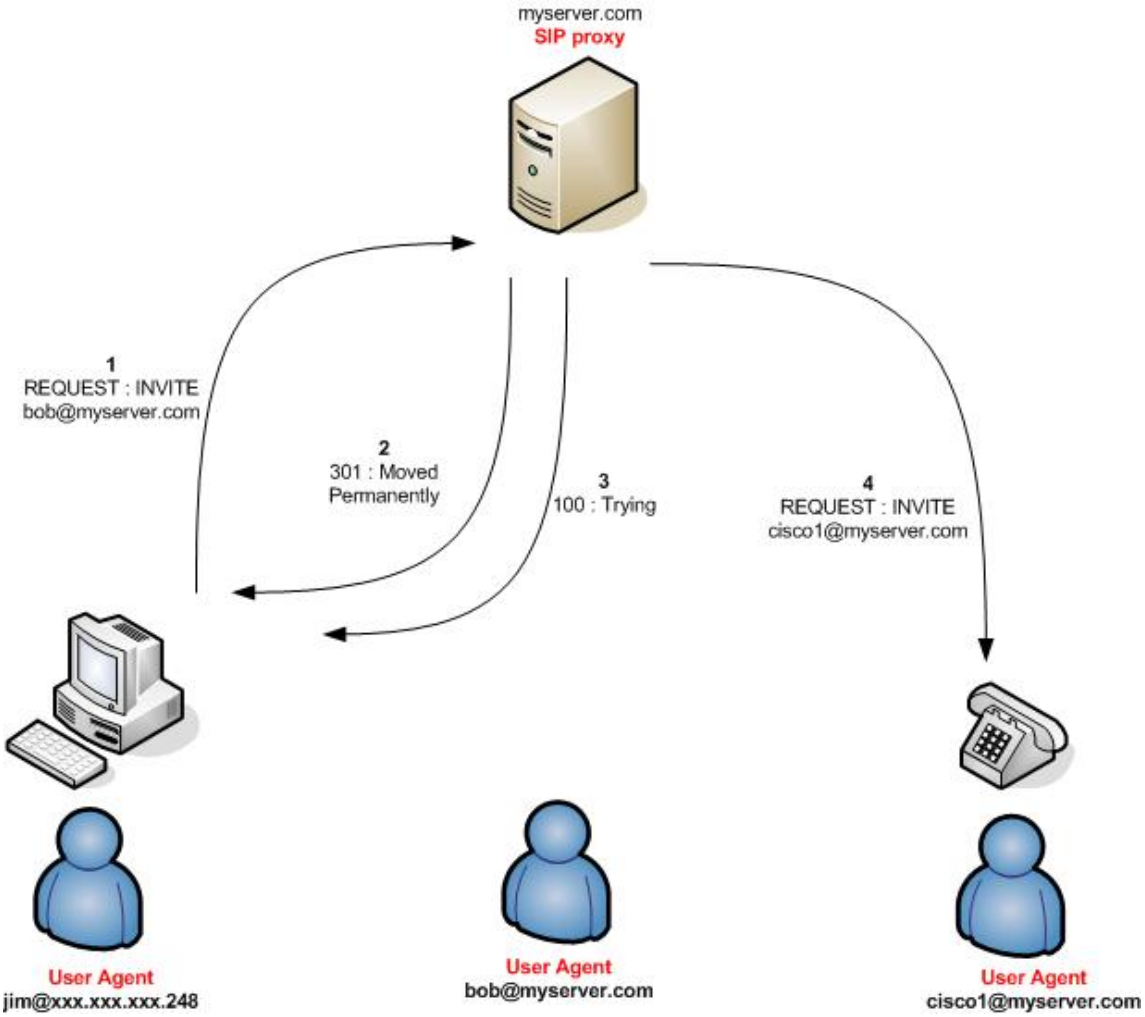

Figure 5.8 : CPL redirection

Figure 5.9 : SIP dialog with CPL

Each user agent will run the implemented program (see appendix A for implementation details) to be context aware. We can reasonably consider that each user uses one User Agent. The context aware service we implemented relies on an OWL database and services around this context source. It gathers context items and uploads CPL scripts. This means that the bottleneck in applying context dependent CPL is the registration of this CPL at the SER server. We need to make sure that the system scale well at its weakest point. Hence we have chosen to measure the performance of this operation for the evaluation of the system. This is described in the next chapter.

6 Evaluation

6.1 Objective and method

As noted in the previous chapter the scaling of a system using context dependent CPL scripts will be limited by the ability to upload these scripts to the relevant incoming SIP proxy. Hence in this section, we will assess the scalability of the SER server when it is using the CPL module. To do so, we will proceed this way:

> **Try to calculate the time required to process a REGISTER message that includes a user's CPL. This duration corresponds to the time interval between the reception of the request on the server's network interface and the server's response being sent on that same interface. The measurements are then made on the server interfaces.**

We will progressively increase the load on the server and then examine the results.

6.2 Experiment A

First, we will measure the time required by the server to answer one request. We used ethereal to measure this duration. We repeated the measurement 10 times in order to avoid irregularities when sending only one request. After 10 measurements, we get an average value of: 4 ms as the time to service each register request.

We can then try to analyze this value more in depth. If we repeat the same experiment with a simple register message, but this time **without** the CPL included in it, we get an average value of 2.5 ms for the response time.

The time difference between the register request with the CPL and the register request without the CPL is mainly due to the database access for storing the CPL script. We can therefore estimate that this access takes : 1.5 ms.

Measurement	register with CPL (ms)	register without CPL (ms)
1	3,37	2,791
2	3,808	1,908
3	4,078	1,983
4	4,401	1,986
5	4,498	2,895
6	4,129	2,958
7	4,129	2,156
8	4,129	2,013
9	4,129	2,917
10	4,129	2,643
Average	**4,08**	**2,425**
Standard deviation	**0,31 (7%)**	**0,45 (18%)**

Figure 6.1 : SER average response time

The standard deviation for the case "register with CPL" shows that all the values are within a reasonable range (standard deviation = 7%). For the case "register without CPL", the standard deviation is much higher (18%) however, the results are not wrong: if we look closer at the values, we can notice that the distribution is bi-modal. The first peak has an average response

time of 2ms and a corresponding standard deviation of 4.5% and the second peak has an average 2.84ms % and a corresponding standard deviation of 4.4%.

	Peak 1 (ms)	Peak 2 (ms)
	1,908	2,791
	1,983	2,895
	1,986	2,958
	2,156	2,917
	2,013	2,643
Average	**2,0092**	**2,8408**
Standard deviation	**0,091 (4,5%)**	**0,127 (4,4%)**

Figure 6.2 : Bi-modal response time of registering without CPL

It would be interesting to see how the response time evolves as the load of the server increases. For this, we need to send several requests to the server in a short period of time. We cannot use the ontology software we made (in java) previously because it is cannot send requests quickly enough to overload the server. In order to do so, we wrote a small C program that opens a UDP socket, reads a SIP register message from a file and then repeatedly sends this message through the socket. The code can be found in the appendix F. The SIP message file was obtained using ethereal to capture an actual SIP registration request. After some initial experiments, we decided to consider the following cases: 50 messages, 100 messages, 150 messages, and 200 messages. We needed at least two measurements in the non-saturated zone, and two in the saturated zone: At least two values in the non saturated zone, so that we can determine the server's service rate, and assess the size of the queue. We found a saturation limit around 120 messages; we could around that limit notice the first unanswered messages. We then took two measurements above this limit to be able to evaluate how fast the server saturates.

6.3 Experiment B

Let x be the number of messages sent and y the average time for the server to answer all x requests. If we send x=50 consecutive register messages in a very short period of time (4 ms), we see that the server is able to answer all the requests. Additionally, we can measure the average response time of the server under such a load, and we get: y=58 ms (compared with 4ms for an isolated request). It is reasonable to get a response time less than 50*4 ms as the server can process several requests in parallel. The list of measurements made for the experiment B is given in the appendix H.

6.4 Experiment C

If we now send x=100 requests in 6.8 ms, the server's average response time increases to y=112 ms, which is approximately the double of 58 ms (of experiment B). The load is now of 100/6.8 = 14,7 requests per millisecond, to compare with 50/4 = 12.5 requests per millisecond for the experiment B. The load has increased, but the response time is **simply** proportional to the number of messages sent. We can therefore conclude that the server is working at 100% of its capacity; i.e. it cannot process messages faster than of rate of roughly 50/58 = 0.86 requests per millisecond.

This means that our software sends packets at a rate which is greater than the rate at which the server can process requests and therefore we can saturate the SER server by filling up its queue.

The list of measurements for the experiment C is given in the appendix H.

6.5 Experiment D

If we repeat the same experiment with 150 requests (sent in 9,4 ms), some requests are not answered. If we repeat the experiment, we get on average 130 answers; as noted at the rate that server processes messages, the queue becomes completely filled and some packets are lost. It is not relevant to calculate a response time.

6.6 Experiment E

The last experiment used 200 messages. Only 150 responses are sent back, which corresponds to a 75% response rate. This means that -on average- a User Agent will have a 93.7 % chances to succeed in registering if it tries twice, and 98.4% if it tries three times. The server is now well into saturation.

6.7 Conclusions and remarks

In experiment B, after 112 ms, the server has completely emptied all its buffers as the last answer has just been sent. Therefore we can assume that if the server gets a series of 50 messages in a row every 112ms, i.e. if we repeat the experiment B every 112ms, the server will not accumulate any delay when answering and will not saturate.

Now, consider now a peak hour, say between 7h55 and 8h05 AM when people arrive at work and are very likely to register with their proxy. This corresponds to a scale of 10 minutes. With the performance of the experiment B, if we consider that a register message corresponds to one user: if the SER server receives 50 messages every 58ms, it will never saturate and therefore it can handle 260 000 users (50*10*60/0,112). This is largely enough to satisfy the needs of a building. This level of load would correspond to 13 000 users registering at a rate of once per 30 seconds or 4 000 users registering at a rate of once per 10 seconds - so that even if user context changes very rapidly such a server could accommodate the number of users in a build or a single site for most companies (see [20] for details of deployment of SIP in a large multination company). This shows that the SER server configured with the CPL module that we used is highly scalable.

We should keep in mind that for this test, SER was not running on a dedicated machine as it would usually be the case for a commercial proxy. We could thoroughly improve the scalability of the server by allocating some more CPU and memory. The hardware and configuration used were described in section 5.

We used for these tests an average sized CPL file. It would have been more revealing to use a longer CPL script because we are then closer to the “worse case”; unfortunately, SER does not yet support complicated scripts (as noted earlier the CPL module is still under development).

Ethereal gets the time stamps from the operating system through the WinPcap library. The precision used in ethereal is one microsecond. As we are measuring a difference between two values, this delay time (required to fetch the time stamp) does not matter (we assume it is constant). The documentation does not provide more information.

7 Conclusions and future work

7.1 Conclusion

In this master thesis work, we studied context aware services and a simple means of using context awareness to enhance a SIP service. We discussed the technical issues raised and examined what is possible with existing software. We can conclude that more technological advances are required, as well as an improved theoretical basis especially for the semantic aspects of calls and call features. Then we focused on context description aspects and assessed OWL's contribution to creating a context aware service. We can conclude that OWL is a major contribution to the context awareness field and that its development will significantly impact context aware service creating. Finally, an implementation of a context aware service was realized using OWL as a context description format and the measurements of its performance in conjunction with an incoming SIP proxy indicate that it is feasible to use for the scenarii which have been described.

This master thesis enabled me to study context awareness and to implement my own context aware service. Context awareness attracted my interest because it aims to have computers serving humans instead of having humans continue to serve them. Ideally, in a context aware world, we would not have to speak the machine's language anymore and only a minimum interaction would be necessary. As can be seen even with these simple scenarii the use of an automatic transition to redirecting calls based upon the user being in a meeting room avoided the user having to manually configure there phone to forward calls to their voice mail.

7.2 Future work

The continuation of this work would be defining a mechanism that would enable a user to easily define the behavior he wants from the proxy server. What can easily be done using today's software is to predefine different context profiles from which the user could select the one that suits his current needs. A more comprehensive approach would define a user interface for collecting personal preferences or use of a learning system to extract such profiles by observing user behavior.

In any case, more CPL scenarii should be written to match more real life situations.

Another extension would be advertising this service. OWL-S seems to be in good position to provide the service advertisement and description, but it would also facilitate the invocation of the service.

References

[1] Dey, A.K.; Abowd, G.D.; Salber, D.
A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications
Source: *Human-Computer Interaction*, v 16, n 2-4, 2001, p 97-166

[2] Myrhaug, H.I.; Whitehead, N.; Goker, A.; Faegri, T.E.; Lech, T.C.
AmbieSense - a system and reference architecture for personalised context-sensitive information services for mobile users
Ambient Intelligence. Second European Symposium, EUSAI 2004. Proceedings (Lecture Notes in Computer Science Vol.3295), 2004, p 327-38

[3] Mark Weiser,
"The Computer for the Twenty-First Century"
Scientific American, pp. 94-10, September 1991.

[4] Chen, Harry; Finin, Tim; Joshi, Anupam
A context broker for building smart meeting rooms
AAAI Spring Symposium - Technical Report, v 4, *Knowledge Representation and Ontologies for Autonomous Systems - Papers from the 2004 AAAI Spring Symposium, Technical Report*, 2004, p 53-60

[5] Pour, G.
Jini for building networked community of devices and services
Proceedings. 34th International Conference on Technology of Object-Oriented Languages and Systems - TOOLS 34, 2000, p 465-74

[6] Jansson, Carl-Gustaf; Jonsson, Martin; Kanter, Theo;
Kilander, Fredrik; Maguire, Gerald; Wei, Li
Adaptive & Context-Aware Services project (**ACAS**); http://psi.verkstad.net/acas, last access January 2006, and Context middleware for adaptive services in heterogeneous wireless networks
IEEE Vehicular Technology Conference, v 61, n 5, *2005 IEEE 61st Vehicular Technology Conference - VTC 2005-Spring Stockholm: Paving the Path for a Wireless Future*, 2005, p 2954-2958

[7] Khedr, Mohamed
Enhancing applicability of context-aware systems using agent-based hybrid inference approach
IEEE Vehicular Technology Conference, v 61, n 5, *2005 IEEE 61st Vehicular Technology Conference - VTC 2005-Spring Stockholm: Paving the Path for a Wireless Future*, 2005, p 2785-2789

[8] Hong, J.I.; Landay, J.A.
An infrastructure approach to context-aware computing
Human-Computer Interaction, v 16, n 2-4, 2001, p 287-303

[9] Castro, Paul; Muntz, Richard
Managing context data for smart spaces
IEEE Personal Communications, v 7, n 5, Oct, 2000, p 44-46

[10] Tao Gu; Hung Keng Pung; Da Qing Zhang
SOCAM, A middleware for building context-aware mobile services
2004 IEEE 59th Vehicular Technology Conference. VTC 2004-Spring (IEEE Cat. No.04CH37514), 2004, pt. 5, p 2656-60 Vol.5

[11] Takahashi, Hideyuki; Suganuma, Takuo; Shiratori, Norio
An agent-based middleware for context-aware ubiquitous services (AMUSE)
Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS, v 1, *Proceedings - 11th International Conference on Parallel and Distributed Systems Workshops, ICPADS 2005*, 2005, p 743-749

[12] Sinnreich H, Johnston A
Internet communications using SIP: Delivering VoIP and multimedia services with Session Initiation Protocol (2001)
Wiley computer publishing,
John Wiley & Sons, 111 River Street, Hoboken, NJ 07030-5774, USA
ISBN: 0-471-41399-2

[13] Lennox J.; X. Wu, H. Schulzrinne
IETF RFC 3380 : Call Processing Language (CPL): A Language for User Control of Internet Telephony (September 2002)
http://www.ietf.org/rfc/rfc3880.txt, last access : January 2006

[14] The World Wide Web Consortium (W3C)
http://www.w3.org, last access, January 2006.

[15] Xiao Hang Wang, Tao Gu, Da Qing
Zhang, Hung Keng Pung
Ontology Based Context Modeling and Reasoning using OWL
Source: http://www.comp.nus.edu.sg/~gutao/gutao_NUS/CNDS2004_gutao.PDF, last access January 2006

[16] SIP Express Router (SER)
www.iptel.org, last access January 2006

[17] H.P. Alesso, C.F. Smith
Developing semantic Web services (2005)
Publisher : AK Peters, 888 Worcester Street, Suite 230, Wellesley, MA 02482, USA
ISBN: 1568812124

[18] Wuthrich, B.
On updates and inconsistency repairing in knowledge bases
Proceedings. Ninth International Conference on Data Engineering (Cat. No.92CH3258-1), 1993, p 608-15

[19] SER configuration
http://www.iptel.org/ser/doc/seruser/seruser.pdf, last access January 2006

[20] Raúl_García
Corporate Wireless IP Telephony
Masters Thesis, Royal Institute of Technology (KTH), 2005.
ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/050802-Raul_Garcia-with-cover.pdf

[21] OWL reasoning
http://owl.man.ac.uk/2003/why/latest/, last access January 2006

[22] Li, W.; Jonsson, M.; Kilander, F.; Jansson, C.G.
Building infrastructure support for ubiquitous context-aware systems
Parallel and Distributed Processing and Applications. Second International Symposium, ISPA 2004. Proceedings (Lecture Notes in Computer Science Vol.3358), 2004, p 509-18

[23] SIP Center
http://www.sipcenter.com/, last access January 2006

[24] Swoogle search engine
http://swoogle.umbc.edu/

Appendices

A. Implementation and deployment

This section explains how to install and configure the SIP Express Router (SER) [16] as a SIP server and how to enable CPL processing. SER is part of the Iptel software associated with the German national research company. It can act as a SIP proxy, registrar, and location server. It has many features (for details see http://www.iptel.org/products/). We will use both the persistence and the CPL modules. Note that the CPL module is not yet considered stable.

a. Environment and Software used

i. Sip Server

This project utilizes a Dell workstation running SuSE 9.3. A SER "distribution" package is available for SuSE, but that binary does not include the CPL module. Therefore, a compilation of SER from its source code was necessary. We used SER version 0.9.4. The sources are available from: ftp://ftp.berlios.de/pub/ser/

ii. Database

The SER architecture requires a database; in this case it uses a MySQL database. In our configuration, the database is located on the same machine as the SER server, but the two servers could be run on different machines. The database provides persistence (necessary to store the CPL scripts) and enables authentication.

iii. User Agent

The project also requires a SIP User Agent. Any SIP UA which is SIP compliant should be satisfactory. We have used Xten's *Xlite* which is the free evaluation of XPro. Xlite is available on Linux and Windows architectures. The binary is available from: http://www.xten.com/index.php?menu=download

iv. Web administration

SER offers an optional web based tool *SERWEB* for the administration of the SIP Express Router. The use of this web interface will be shown in section 5.2.5. SERWEB can be downloaded from: ftp://ftp.berlios.de/pub/ser/latest/contrib/.

v. CPL management

CPLEd can be used to edit and upload CPL scripts to the SIP server. It can validate the scripts against any provided DTD. It offers two ways to upload scripts, via HTTP and via SIP. HTTP is not fully operational at this time , thus we have used SIP. CPLEd can be downloaded from: http://developer.berlios.de/projects/cpled/

CPLEd's code has been modified to provide a java CPL uploader that takes context information as input, then chooses which CPL script to upload on the server. The modifications to make are described in 5.5. The classes to add to the CPL Ed source code are provided in the appendix G.

b. Installing SER

i. Required libraries

First, some libraries are required:

- libmysqlclient-dev
- libpq-dev
- flex, sed and tr (for compiling)
- libxml2 for CPL support

After installing these libraries you can now begin to compile the SER sources.

ii. Compilation

First you need to untar the sources:

```
tar -xvzf ser-0.9.4_src.tar.gz
```

then you need to modify the makefile to enable CPL-C - which requires the additional modules tm and mysql: the tm, cpl-c, and mysql modules should be transferred from the excluded modules part to the included modules.

The resulting makefile should include the following lines:

```
# if not set on the cmd. line or the env, exclude this modules:
exclude_modules?=   cpl ext extcmd \
                            postgres snmp \
                            im \
                            jabber \
                            auth_radius group_radius uri_radius avp_radius \
                            pa

# always exclude the CVS dir
override exclude_modules+= CVS $(skip_modules)

#always include this modules
```

```
include_modules?= mysql tm cpl-c
```

You can now compile the whole package and install it:

```
make all include_modules="mysql tm cpl-c"
make install include_modules="mysql tm cpl-c"
```

The tm (transaction module) is required for CPL.

These commands will install

- the configuration files under /usr/local/etc/
- the executables under /usr/local/sbin/

It is also possible to place these files in other locations, but these are the default locations. If you want to have SER installed elsewhere for instance :

- the configuration files under /etc/
- the executables under /sbin/

You must then set the following:

```
make prefix=  all include_modules="mysql tm cpl-c"
make prefix=  install include_modules="mysql tm cpl-c"
```

iii. Server configuration

1. Launching and stopping the server

To start the server, enter the following command (depending on the path specified for the executables):

```
/usr/local/sbin/ser
```

The default port used is 5060.

A nice way of stopping and starting the server is to use a service file (adapted from another server). An exemple of such a file is provided in appendix B:

Thus to start and stopthe server type, you only need to enter:

```
/etc/init.d/ser <start|stop>
```

2. serctl tool

serctl is a tool provided with SER. It helps monitoring and configuring the server. To use it, define this global variable:

```
export SIP_DOMAIN="localhost"
```

It is a useful tool for managing the SER server. Features comprise users management (adding, removing users, changing passwords)…. see section 5.2.4 (user provisioning)

For instance, to see in real-time the server's activity, type :

```
serctl moni
```

To display all the serctl tool capabilities, type:

```
sertcl -help
```

3. MySQL database

Start the MySQL server:

```
/etc/init.d/mysql start
```

Run the *ser_mysql.sh* script located at /usr/local/sbin/
It will ask you for the root password of the database.

The script will create the SER database that should looks like this (except that it does not yet have the cpl table…):

	Table	Action	Enregistrements	Type	Interclassement	Taille
☐	acc		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	active_sessions		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	admin_privileges		2	MyISAM	latin1_swedish_ci	3,1 Ko
☐	aliases		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	calls_forwarding		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	config		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	cpl		1	MyISAM	latin1_swedish_ci	3,4 Ko
☐	domain		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	event		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	grp		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	location		3	MyISAM	latin1_swedish_ci	3,4 Ko
☐	missed_calls		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	pending		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	phonebook		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	preferences_types		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	reserved		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	server_monitoring		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	server_monitoring_agg		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	silo		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	speed_dial		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	subscriber		5	MyISAM	latin1_swedish_ci	4,9 Ko
☐	trusted		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	uri		4	MyISAM	latin1_swedish_ci	3,2 Ko
☐	usr_preferences		0	MyISAM	latin1_swedish_ci	1,0 Ko
☐	version		24	MyISAM	latin1_swedish_ci	1,5 Ko
	25 table(s)	Somme	39	--	latin1_swedish_ci	38,6 Ko

Figure 5.1 : Database preview

A database user called ser is also created. The corresponding password is “heslo”. This default password should be changed, for instance using the MySQL prompt line:

```
shell  >mysql –u root -p
myslq > SET PASSWORD FOR 'ser'@'localhost' = PASSWORD('newpwd');
myslq > SET PASSWORD FOR ''@'%' = PASSWORD('newpwd');
```

To create the CPL table, enter:

```
USE ser;
CREATE TABLE cpl ( user VARCHAR(50) NOT NULL PRIMARY KEY, cpl_xml BLOB, cpl_bin BLOB,
UNIQUE (user));
```

The cpl_xml field will be used to store the CPL XML script and the cpl_bin field will store it in a SER executable format. The VARCHAR(50) corresponds to the user name. We dimensioned it to 50 characters to make sure all possible names could fit in.

4. SER configuration file

SER has one main configuration file. It is called ser.cfg and located at /usr/local/etc/ser/. A sample configuration is provided and is called ser.cfg.sample. The configuration file has four parts:

 # Global Parameters

define the deamon’s configuration: it specifies the debug level, if the deamon should fork, the port it is bound to,...

 # External Module Loading

The syntax to load a module is :

```
loadmodule "path_to_module"
```

You should make sure that the CPL module is loaded :

```
loadmodule "/usr/local/lib/ser/modules/cpl-c.so"
```

There is a specific order for loading modules:
The cpl-c.so module should **not** be loaded before the the sl.so, tm.so, and usrloc.so modules, as it depends upon these modules.

Uncomment the module corresponding to mysql. You should also uncomment the following modules used for authentication:

```
loadmodule "/usr/local/lib/ser/modules/mysql.so"
loadmodule "/usr/local/lib/ser/modules/auth.so"
loadmodule "/usr/local/lib/ser/modules/auth_db.so"
```

 # Modules Parameters

This section specifies the parameters for the modules with the following syntax:

```
modparam("module","parameter","value_of_parameter")
```

The CPL module requires some parameters to be set. It is mandatory to specify them is the configuration file. One could adapt the values shown below (to fit your own installation).

First you need to specify the location of the database as well as the username and password to be used to connect to the database:

```
modparam("cpl_c","cpl_db","mysql://user:passwd@host/database")
```

Next you need to specify where the XML DTD for the CPL is to be found, this is the CPL DTD path:

```
modparam("cpl_c","cpl_dtd_file","/etc/ser/cpl-06.dtd")
```

Finally you need to indicate the name of the CPL table in the database:

```
modparam("cpl_c","cpl_table","cpltable")
```

Other parameters can also be specified, but are not compulsory; for details, look at the associated README file in the 'cpl-c' folder in the sources.

The ser.cfg used in this example is provided in the appendix C.

 # Routing Blocks

This part of the configuration is the most complex. SER uses its own routing language. This part specifies the "routing logic" of the server, i.e. how it should behave depending on the request it receives for each user.

The code is decomposed into "route blocks":

The first one (route[0]) is the first to be invoked upon reception of a message. It is usually used for dispatching requests to other route blocks. However, it can also take care of registration.

This language allows regular expressions and hence the server must parse requests and can adjust its behavior. A complete description of this language can be found in [19].

iv. Users provisioning

To provision users, i.e., add a new user, you type:

```
serctl add userName userPassword userEmailAdress
```

This command requires entering the SER database password. The password should have already been changed! Another way to do this is to use the web interface -as described in the following section.

v. SERWeb administration

Using the web interface requires installing a web server. The application is written in PHP. The PHP Extension and Application Repository (PEAR) is also required; it is available from http://pear.php.net/.

To install SERWeb, first untar the package and move the php files to a directory accessible to you web server (with Apache, it could be for example /srv/www/htdocs/).

It is wise to create an alias in your httpd.conf file, thus avoiding libraries and scripts being accessible. The Alias directive allows documents to be stored in the local filesystem other than under the DocumentRoot. URLs with a path beginning with */serweb/* will be mapped to local files beginning with *"/srv/www/htdocs/serweb/html/"*.

```
Alias /serweb/ "/srv/www/htdocs/serweb/html/"
```

Allow only the html folder to be accessible and not other folders such as `/srv/www/htdocs/serweb/scripts/` for example.

Open the file '`config_data_layer.php`' and configure it according to your database. Enter ser as a user, the corresponding password and the address and port to be used.

Make sure that the web server has the necessary permissions to write in /var/log/serweb

```
touch /var/log/serweb
chown www-data /var/log/serweb
```

Check also that PHP configuration file php.ini has the short_open_tag parameter set to 'on'

```
short_open_tag = on
```

The *short_open_tag* Boolean tells whether the short form (`<? ?>`) of PHP's open tag should be allowed.

Modifications of the php configuration ("include tags") will be required to enable PHP to find the PEAR libraries. The easiest method is to not modify the PHP pages, but simply move the libraries into the html directory.

One can access the admin interface with any web browser at:

```
http://<your-host>/<your-install-dir>/admin/index.php
```

The user is admin@your-host and the password is what you set earlier.

vi. User Agent configuration

Figures 5.2 and Figure 5.3 show respectively the Xten Xlite user agent and its configuration.

Figure 5.2 : Xten User Agent

Figure 5.3 : User Agent Configuration

In the configuration menu:

- you should set enable to YES
- the username is the same as the one provided to the serctl tool
- enter the same password as well
- enter the realm (domain) of your SIP server
- fill in the SIP Proxy field, note that you should specify the port, the default is 5060
- enter the SIP Proxy address of your outbound proxy

If you are logged in, you should see the message “logged in” on the top left of your screen (as shown in figure 5.2). In the case of the Xten Xlite user interface, you can quickly switch from numeric typing to letter using the space key, as needed to fill in the configuration menu.

c. CPLEd modifications

We will describe here the steps required to modify CPLEd (CPL editor) in order to obtain the software we used.

The CPLEd is a java application. It comes with the 4 following packages:

- network (to manage network connections)
- tree (to manage CPL scripts)
- gui (the graphical interface)
- util (all kind of tools, for other packages)

We can erase the gui package as we will not need a graphical interface and add the two following packages:

- context (upload of CPL)
- owl (to manage OWL files)

The classes to add within these packages are provided in the appendix G.

B. SER server startup and stop file

```
#! /bin/sh
# Copyright (c) 2002 Frauenhofer Gesellschaft FOKUS, Germany.
# All rights reserved.
#
# Author: Nils Ohlmeier <ohlmeier@fokus.fhg.de>
#
# /etc/init.d/ser
#
### BEGIN INIT INFO
# Provides:          sip
# Required-Start:    $network
# X-UnitedLinux-Should-Start: $network
# Required-Stop:     $network
# X-UnitedLinux-Should-Stop: $network
# Default-Start:     3 5
# Default-Stop:      0 1 2 6
# Short-Description: SIP Express Router
# Description:       Start SER and provide the routing of SIP requests.
### END INIT INFO


# Check for missing binaries (stale symlinks should not happen)
SER_BIN=/usr/local/sbin/ser
test -x $SER_BIN || exit 5

# Check for existence of needed config file and read it
SER_CONFIG=/usr/local/etc/ser/ser.cfg
test -r $SER_CONFIG || exit 6
#. $FOO_CONFIG

# Shell functions sourced from /etc/rc.status:
#      rc_check         check and set local and overall rc status
#      rc_status        check and set local and overall rc status
#      rc_status -v     ditto but be verbose in local rc status
#      rc_status -v -r  ditto and clear the local rc status
#      rc_status -s     display "skipped" and exit with status 3
#      rc_status -u     display "unused" and exit with status 3
#      rc_failed        set local and overall rc status to failed
#      rc_failed <num>  set local and overall rc status to <num>
#      rc_reset         clear local rc status (overall remains)
#      rc_exit          exit appropriate to overall rc status
#      rc_active        checks whether a service is activated by symlinks
#      rc_splash arg    sets the boot splash screen to arg (if active)
. /etc/rc.status

# Reset status of this service
```

```
rc_reset

# Return values acc. to LSB for all commands but status:
# 0       - success
# 1       - generic or unspecified error
# 2       - invalid or excess argument(s)
# 3       - unimplemented feature (e.g. "reload")
# 4       - user had insufficient privileges
# 5       - program is not installed
# 6       - program is not configured
# 7       - program is not running
# 8--199  - reserved (8--99 LSB, 100--149 distrib, 150--199 appl)
#
# Note that starting an already running service, stopping
# or restarting a not-running service as well as the restart
# with force-reload (in case signaling is not supported) are
# considered a success.

case "$1" in
    start)
        echo -n "Starting SIP Express Router "
        ## Start daemon with startproc(8). If this fails
        ## the return value is set appropriately by startproc.
        startproc $SER_BIN -f $SER_CONFIG

        # Remember status and be verbose
        rc_status -v
        ;;
    stop)
        echo -n "Shutting down SIP Express Router "
        ## Stop daemon with killproc(8) and if this fails
        ## killproc sets the return value according to LSB.

        killproc -TERM $SER_BIN

        # Remember status and be verbose
        rc_status -v
        ;;
    restart)
        ## Stop the service and regardless of whether it was
        ## running or not, start it again.
        $0 stop
        $0 start

        # Remember status and be quiet
        rc_status
        ;;
    force-reload)
        ## Signal the daemon to reload its config. Most daemons
        ## do this on signal 1 (SIGHUP).
        ## If it does not support it, restart.
```

```
        echo -n "Reload service SIP Express Router "
        ## Otherwise:
        $0 stop  &&  $0 start
        rc_status
        ;;
    reload)
        ## Like force-reload, but if daemon does not support
        ## signaling, do nothing (!)

        ## Otherwise if it does not support reload:
        rc_failed 3
        rc_status -v
        ;;
    status)
        echo -n "Checking for service SIP Express Router "
        ## Check status with checkproc(8), if process is running
        ## checkproc will return with exit status 0.

        # Return value is slightly different for the status command:
        # 0 - service up and running
        # 1 - service dead, but /var/run/  pid  file exists
        # 2 - service dead, but /var/lock/ lock file exists
        # 3 - service not running (unused)
        # 4 - service status unknown :-(
        # 5--199 reserved (5--99 LSB, 100--149 distro, 150--199 appl.)

        # NOTE: checkproc returns LSB compliant status values.
        checkproc $SER_BIN
        # NOTE: rc_status knows that we called this init script with
        # "status" option and adapts its messages accordingly.
        rc_status -v
        ;;
    *)
        echo "Usage: $0 {start|stop|status|restart}"
        exit 1
        ;;
esac
rc_exit
```

C. SER configuration file: ser.cfg

```
#
# simple quick-start config script
#

# ----------- global configuration parameters ------------------------

debug=6         # debug level (cmd line: -dddddddddd)
#fork=yes
#log_stderror=no  # (cmd line: -E)

/* Uncomment these lines to enter debugging mode
debug=7
fork=no
log_stderror=yes
*/

check_via=no      # (cmd. line: -v)
dns=no            # (cmd. line: -r)
rev_dns=no        # (cmd. line: -R)
#port=5060
#children=4
fifo="/tmp/ser_fifo"

# ------------------ module loading ----------------------------------

# Uncomment this if you want to use SQL database
loadmodule "/usr/local/lib/ser/modules/mysql.so"

loadmodule "/usr/local/lib/ser/modules/sl.so"
loadmodule "/usr/local/lib/ser/modules/tm.so"
loadmodule "/usr/local/lib/ser/modules/rr.so"
loadmodule "/usr/local/lib/ser/modules/maxfwd.so"
loadmodule "/usr/local/lib/ser/modules/usrloc.so"
loadmodule "/usr/local/lib/ser/modules/registrar.so"
loadmodule "/usr/local/lib/ser/modules/textops.so"

# Uncomment this if you want digest authentication
# mysql.so must be loaded !
loadmodule "/usr/local/lib/ser/modules/auth.so"
loadmodule "/usr/local/lib/ser/modules/auth_db.so"

# ----------------- setting module-specific parameters ---------------

# -- usrloc params --

#modparam("usrloc", "db_mode",   0)

# Uncomment this if you want to use SQL database
# for persistent storage and comment the previous line
modparam("usrloc", "db_mode", 2)

# -- auth params --
# Uncomment if you are using auth module
#
modparam("auth_db", "calculate_ha1", yes)
#
# If you set "calculate_ha1" parameter to yes (which true in this config),
# uncomment also the following parameter)
```

```
#
modparam("auth_db", "password_column", "password")

# -- rr params --
# add value to ;lr param to make some broken UAs happy
modparam("rr", "enable_full_lr", 1)


modparam("usrloc", "db_url","mysql://ser:heslo@130.237.15.247/ser")

#---CPL-C Parameter

loadmodule "/usr/local/lib/ser/modules/cpl-c.so"

modparam("cpl-c","cpl_db","mysql://ser:heslo@130.237.15.247/ser")
modparam("cpl-c","cpl_table","cpl")
modparam("cpl-c","cpl_dtd_file","/usr/local/etc/ser/cpl.dtd")
#modparam("cpl-c","log_dir","/var/log/ser/cpl")
#modparam("cpl-c","proxy_recurse",2)
#modparam("cpl-c","proxy_route",0)
#modparam("cpl-c","nat_flag",6)
#modparam("cpl-c","lookup_domain","location")
#cpl_run_script("incoming","force_statefull");

# -------------------------  request routing logic -------------------

# main routing logic

route{
if (uri==myself) {
 if (method=="REGISTER")
  {
     # Uncomment this if you want to use digest authentication
     #if (!www_authorize("ccsser1.wireless.kth.se", "subscriber")) {
www_challenge("ccsser1.wireless.kth.se", "0"); break;};
     # handle REGISTER messages with CPL script
     cpl_process_register();
     save("location");
     break;
  };
# process INVITE
 if (method=="INVITE")
   {
     # cpl interpreter requires a pre-created transaction for the processed
INVITE
     if (!t_newtran()) {
      # it's a retransmission
      break;
     };

     if (!cpl_run_script("incoming","force_stateful")) {
      #script execution failed
      t_reply("500", "CPL script execution failed");
     };


     # we get here only if the CPL interpreter decided that server
     # should follow it default behavior
     # mark that there is already a created transaction for current INVITE
     setflag(3);
   };
   # native SIP destinations are handled using our USRLOC DB
   if (!lookup("location")) {
```

```
     sl_send_reply("404", "Not Found");
     break;
   };
 };
# forward current uri now
 if (!isflagset(3)) {
   # build a new fresh transaction and forward
   if (!t_relay()) {
    sl_reply_error();
   };
  }
  else {
   # transaction exists -> do just forward
   if (!t_forward_nonack_uri()) {
    sl_reply_error();
   };
  };
 }
```

D. CPL scripts

callRedirectUnconditional

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE cpl PUBLIC '-//IETF//DTD RFCxxxx CPL 1.0//EN' 'cpl.dtd'>

<cpl>
 <incoming>
  <location url="sip:lisa@myhost.com">
   <redirect permanent="yes" />
  </location>
 </incoming>
</cpl>
```

screeningRedirectBasedOnCaller

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE cpl PUBLIC '-//IETF//DTD RFCxxxx CPL 1.0//EN' 'cpl.dtd'>

<cpl>
 <incoming>
  <address-switch field="origin" subfield="user">
   <address is="jim">
    <location url="sip:lisa@myhost.com">
     <redirect permanent="yes" />
    </location>
   </address>
  </address-switch>
 </incoming>
</cpl>
```

screeningRejectionBasedOnCaller

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE cpl PUBLIC '-//IETF//DTD RFCxxxx CPL 1.0//EN' 'cpl.dtd'>

<cpl>
 <incoming>
  <address-switch field="origin" subfield="user">
   <address is="jim">
    <reject status="reject" reason="younesRejection" />
   </address>
  </address-switch>
 </incoming>
</cpl>
```

screeningRejectionBasedOnHost

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE cpl PUBLIC '-//IETF//DTD RFCxxxx CPL 1.0//EN' 'cpl.dtd'>

<cpl>
 <incoming>
  <address-switch field="origin" subfield="host">
   <address subdomain-of="domain.com">
    <reject status="reject" reason="younesRejection" />
   </address>
  </address-switch>
 </incoming>
</cpl>
```

E. OWL context ontology

```
<?xml version="1.0"?>
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:yo="http://213.100.49.8/owl/personCentered#">
        <rdf:Description rdf:about="urn:http://213.100.49.8/personCentered">
                <rdf:type>
                        <rdf:Description
rdf:about="http://www.w3.org/2002/07/owl#Ontology"/>
                </rdf:type>
        </rdf:Description>
        <owl:Class rdf:ID="Person">
                <rdf:type>
                        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
                </rdf:type>
        </owl:Class>
        <owl:Class rdf:ID="User">
                <rdf:type>
                        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
                </rdf:type>
                <rdfs:subClassOf>
                        <rdf:Description rdf:about="#Person"/>
                </rdfs:subClassOf>
        </owl:Class>
        <owl:Class rdf:ID="userContext">
                <rdf:type>
                        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
                </rdf:type>
        </owl:Class>
        <rdf:Description rdf:about="urn:hasName">
                <rdf:type>
                        <rdf:Description
rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
                </rdf:type>
                <rdfs:domain>
                        <rdf:Description rdf:about="#Person"/>
                </rdfs:domain>
                <rdfs:range>
                        <rdf:Description
rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
                </rdfs:range>
        </rdf:Description>
        <rdf:Description rdf:about="urn:hasContacts">
                <rdf:type>
                        <rdf:Description
rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
                </rdf:type>
```

```
                <rdfs:domain>
                        <rdf:Description rdf:about="#User"/>
                </rdfs:domain>
                <rdfs:range>
                        <rdf:Description rdf:about="#Person"/>
                </rdfs:range>
        </rdf:Description>
        <rdf:Description rdf:about="urn:hasContext">
                <rdf:type>
                        <rdf:Description
rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
                </rdf:type>
                <rdfs:domain>
                        <rdf:Description rdf:about="#User"/>
                </rdfs:domain>
                <rdfs:range>
                        <rdf:Description rdf:about="#userContext"/>
                </rdfs:range>
        </rdf:Description>
        <rdf:Description rdf:about="urn:hasLocation">
                <rdf:type>
                        <rdf:Description
rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
                </rdf:type>
                <rdfs:domain>
                        <rdf:Description rdf:about="#userContext"/>
                </rdfs:domain>
                <rdfs:range>
                        <rdf:Description
rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
                </rdfs:range>
        </rdf:Description>
        <rdf:Description rdf:about="urn:hasTask">
                <rdf:type>
                        <rdf:Description
rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
                </rdf:type>
                <rdfs:domain>
                        <rdf:Description rdf:about="#userContext"/>
                </rdfs:domain>
                <rdfs:range>
                        <rdf:Description
rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
                </rdfs:range>
        </rdf:Description>
</rdf:RDF>
```

F. Evaluation program

```
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include <winsock.h>

int main(int argc, char *argv[])
{
  // Winsock initialisation
  WSADATA wsaData ;
  int res ;
  res = WSAStartup(MAKEWORD(2 ,0), &wsaData) ;

  // socket creation
  SOCKET m_socket;
  //m_socket = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );
  m_socket = socket( AF_INET, SOCK_DGRAM, IPPROTO_UDP );
  if ( m_socket == INVALID_SOCKET ) {
    printf( "Error at socket(): %ld\n", WSAGetLastError() );
    WSACleanup();
    return;
  }

  struct sockaddr_in clientService;
  clientService.sin_family = AF_INET;
  clientService.sin_addr.s_addr = inet_addr( "130.237.15.247" );
  clientService.sin_port = htons( 5060 );

  /*
  if ( connect( m_socket, (SOCKADDR*) &clientService, sizeof(clientService)
) == SOCKET_ERROR) {
    printf( "Failed to connect.\n" );
    WSACleanup();
    return;
  }*/

  int bytesSent;
  int bytesRecv = SOCKET_ERROR;
  char sendbuf[32] = "Client: Sending data.";
  char recvbuf[32] = "";

  /*
  bytesSent = send( m_socket, sendbuf, strlen(sendbuf), 0 );
  printf( "Bytes Sent: %ld\n", bytesSent );
  */

  char mega [1000];
  readPacket( mega );

  //int PASCAL sendto(SOCKET,const char*,int,int,const struct
sockaddr*,int);
  //int check = sendto( m_socket, sendbuf, strlen(sendbuf), 0, (struct
sockaddr*)&clientService, sizeof(clientService)  ) ;
  /*******************************  k is the number of sent packets
*************************/
  int k;
```

```
  for( k=1 ; k<2 ; k++){
    int check = sendto( m_socket, mega, strlen(mega), 0, (struct
sockaddr*)&clientService, sizeof(clientService)  ) ;
  }

  system("PAUSE");
  return 0;

}


int readPacket( char* ptrBuf ){

    printf("essai ici");
    FILE *fileToRead;
    char szBuffer;

    /* Ouverture d'un fichier */
   fileToRead = fopen( "sipmessage", "rb");
   if( fileToRead == NULL ) {
       printf("\nErreur en lecture de %s\n", "fwrite.ex");
       system("PAUSE");
       exit(2);
   }

   int i=0;

   while (  !feof(fileToRead) ) {
         fread(&szBuffer, 1, 1, fileToRead);
         ptrBuf[i] = szBuffer;
         i++;
   }

   fclose(fileToRead);

}
```

G. CPLEd added classes

Launcher.java

```
package CPLEd.context;

import java.io.FileInputStream;

import CPLEd.network.TransportCPL;
import CPLEd.owl.UserData;
import CPLEd.tree.CPLTree;
import CPLEd.util.Global;

public class Launcher {

      public void run( UserData ud, String pathToDTD, String pathToScript,
String passwd, String sipURL){
            Global global = new Global();

            try{
                  // CPL script validation
                  global.setCPLTree( new CPLTree(global) );
                  global.setProp("DTDlocation", pathToDTD );
                  global.getCPLTree().loadCPL(new
FileInputStream(pathToScript));

                  // CPL script upload
                  global.setProp("user", ud.getUserName() );
                  global.setProp("passwd", passwd );
                  global.setProp("SipURL", sipURL );

                  TransportCPL cpl_uploader = new TransportCPL( global,
"upload", "SIP" );
                  cpl_uploader.executeOperation();
                  //cpl_uploader.executeOperation();

            }catch(Exception e){
                  e.printStackTrace();
            }
      }

}
```

Inter.java

```
package CPLEd.owl;

import CPLEd.context.*;


public class Inter {

	String pathToDTD = "file:E:/prog/scriptsCPL/cpl.dtd";
	String passwd = "ares";
	String SipURL = "ccsser1.wireless.kth.se";
	String pathToScript = "";
	UserData userData;


	public static void main(String[] args) {
		Inter inter = new Inter();
		inter.userData = new UserData();
		inter.readOWL(inter.userData);
		inter.manageCPL();
	}

	private void readOWL(UserData userData){
		ParsingOWL parser = new ParsingOWL();
		//parser.validateOWL();
		parser.parseOWL(userData);
	}

	private void manageCPL(){
		Launcher launcher = new Launcher();
		this.pathToScript = determineScriptToUse(this.userData);
		launcher.run( userData, pathToDTD, pathToScript, passwd, SipURL
);
	}

	private String determineScriptToUse( UserData userData ){

		String myPathToScript = "";
		String task = userData.getUserTask();
		String location = userData.getUserLocation();

		if ( task.equals("meeting") || location.equals("grimeton") ){
			myPathToScript =
"E:/prog/scriptsCPL/workingScriptsIncoming/callRedirectUnconditional.xml";
		}else if ( task.equals("work") && location.equals("office") ){
			myPathToScript =
"E:/prog/scriptsCPL/workingScriptsIncoming/screeningRedirectBasedOnCaller.x
ml";
		}else if ( location.equals("home") || task.equals("relaxing")
){
			myPathToScript =
"E:/prog/scriptsCPL/workingScriptsIncoming/screeningRejectionBasedOnHost.xm
l";
		}
		return myPathToScript;
	}

}
```

ParsingOWL.java

```
package CPLEd.owl;

import org.xml.sax.*;
import javax.xml.parsers.*;
import java.io.*;

import java.util.Enumeration;
import java.util.Vector;

import com.bbn.semweb.owl.vowlidator.ValidatorAPI;
import com.bbn.semweb.owl.vowlidator.indications.Indication;

public class ParsingOWL {

	public void validateOWL(){
		try{
			ValidatorAPI api = new ValidatorAPI();
			// Load models/URI to parse against
			//api.addURL("http://www.daml.org/2003/02/usps/usps-
ont.owl", true);

	//api.addURL("http://172.16.5.78/owl/userContextOntology", true);
				api.addURL("file:///E:/prog/owl/userContextOntology.owl",
true);
				//api.addURL("http://www.w3.org/2002/07/owl", true);

				// Create RDF Model to validate
				//String url =
"http://www.daml.org/2003/02/usps/state.owl";
				//String url =
"http://www.daml.org/2003/01/periodictable/PeriodicTable";
				//String url = "http://213.100.49.8/owl/bobMeeting.owl";
				String url = "file:///E:/prog/owl/bobMeeting.owl";

				// Validate Model
				Vector indications = api.validateURL(url);

				// Print out Indications
				System.out.println("\nVALIDATION RESULTS\n");
				if (indications.size() == 0) {
				    System.out.println("No Indications");
				} else {
				    for (Enumeration e = indications.elements();
e.hasMoreElements(); ) {
				      Indication i = (Indication)e.nextElement();
				      System.out.println(i.toString());
				    }
				}

		}catch(Exception e){
			e.printStackTrace();
		}
	}

	public void parseOWL( UserData userData ){
		try{
		   // création d'une fabrique de parseurs SAX
		   SAXParserFactory fabrique = SAXParserFactory.newInstance();
		   // création d'un parseur SAX
		   SAXParser parseur = fabrique.newSAXParser();
```

```
                // lecture d'un fichier XML avec un DefaultHandler
                File fichier = new File("E:/prog/owl/bobRelaxing.owl");

                UserDataHandler handler = new UserDataHandler(userData);
                parseur.parse(fichier, handler);

            }catch(ParserConfigurationException pce){
                System.out.println("Erreur de configuration du parseur");
                System.out.println("Lors de l'appel à newSAXParser()");
            }catch(SAXException se){
                System.out.println("Erreur de parsing");
                System.out.println("Lors de l'appel à parse()");
            }catch(IOException ioe){
                System.out.println("Erreur d'entrée/sortie");
                System.out.println("Lors de l'appel à parse()");
            }
        }


}
```

UserData.java

```
package CPLEd.owl;

public class UserData {

      String userName = "";
      String userLocation = "";
      String userTask = "";

      public String getUserLocation() {
            return userLocation;
      }
      public void setUserLocation(String userLocation) {
            this.userLocation = userLocation;
      }
      public String getUserName() {
            return userName;
      }
      public void setUserName(String userName) {
            this.userName = userName;
      }
      public String getUserTask() {
            return userTask;
      }
      public void setUserTask(String userTask) {
            this.userTask = userTask;
      }
}
```

UserDataHandler.java

```
package CPLEd.owl;

import org.xml.sax.*;
import org.xml.sax.helpers.*;


//implements ContentHandler
public class UserDataHandler extends DefaultHandler{

      boolean locationBoolean;
      boolean taskBoolean;
      String userName;
      String location;
      String task;
      UserData user;

      public void startElement(String nameSpaceURI, String localName,
String rawName, Attributes attributs) throws SAXException {
        if ( rawName.equals("acas:User")){
            //System.out.println("YOUNES COMMENT : we are in the user
tag");
            for (int index = 0; index < attributs.getLength(); index++) {
// on parcourt la liste des attributs
                  // we read the name of the user from the attribute rdf:ID
                  if ( (attributs.getQName(index)).equals("rdf:ID") ){
                        this.userName = attributs.getValue(index);
                  }
            }
        }else if ( rawName.equals("acas:hasLocation") ) {
            locationBoolean = true;
        }else if ( rawName.equals("acas:hasTask") ) {
            taskBoolean = true;
        }
      }

      public void characters(char[] ch, int start, int end) throws
SAXException {
            if ( locationBoolean ){
                  location = new String( ch, start, end );
            }else if ( taskBoolean ){
                  task = new String( ch, start, end );
            }
      }

      public void endElement(String nameSpaceURI, String localName, String
rawName) throws SAXException {
            if( locationBoolean){
                  locationBoolean = false;
            }
            if ( taskBoolean ){
                  taskBoolean = false;
            }
    }

      public UserDataHandler ( UserData userData ){
            super();
            this.user = userData;
      }

      public void startDocument() throws SAXException {
```

```
        //System.out.println("Debut de l'analyse du document");
    }

    public void endDocument() throws SAXException {
        //System.out.println("userName: "+userName);
        //System.out.println("Location: "+location);
        //System.out.println("Task    : "+task);
            user.setUserName(userName);
            user.setUserTask(task);
            //user.set
            System.out.println("Fin de l'analyse du document" );
    }
}
```

H. Experiment B and C measurements

Experiment B, all times in µs

Request number	50 register with CPL
1	39
2	120
3	245
4	257
5	370
6	383
7	495
8	508
9	619
10	744
11	868
12	994
13	1004
14	1244
15	1260
16	1369
17	1618
18	1743
19	1754
20	1868
21	1879
22	1993
23	2117
24	2242
25	2367
26	2377
27	2618
28	2629
29	2743
30	2867
31	2877
32	2994
33	3117
34	3130
35	3242
36	3253
37	3367
38	3378
39	3492
40	3503
41	3618
42	3629
43	3743
44	3755
45	3868
46	3878
47	3992

48	4002
49	4242
AVERAGE:	**2212**

response number	**50 register with CPL**
1	27218
2	27342
3	29677
4	29911
5	39552
6	42196
7	51525
8	51655
9	51789
10	51879
11	54019
12	59649
13	59803
14	78045
15	78227
16	78601
17	78746
18	88579
19	93321
20	93529
21	95849
22	96021
23	103603
24	110539
25	112886
26	113056
27	131100
28	134259
29	137919
30	141140
31	142388
32	143969
33	149895
34	151357
35	157734
36	164119
37	165079
38	173998
39	177463
40	177600
41	182618
42	182742
43	187291
44	189827
45	190130
46	193019
47	193645
48	193848
49	194910
AVERAGE:	**115373**

Experiment C, all times in μs

request number	100 register with CPL
1	0
2	31
3	126
4	139
5	250
6	263
7	375
8	388
9	500
10	624
11	749
12	760
13	875
14	886
15	999
16	1010
17	1250
18	1263
19	1272
20	1374
21	1499
22	1511
23	1519
24	1624
25	1749
26	1760
27	1873
28	1884
29	1998
30	2008
31	2123
32	2248
33	2259
34	2373
35	2382
36	2498
37	2509
38	2623
39	2634
40	2748
41	2759
42	2873
43	2884
44	2998
45	3008
46	3123
47	3133
48	3248
49	3373
50	3382
51	3498
52	3509

53	3622
54	3747
55	3757
56	3764
57	3873
58	3997
59	4010
60	4122
61	4133
62	4247
63	4258
64	4372
65	4497
66	4511
67	4621
68	4634
69	4746
70	4756
71	4871
72	4882
73	4995
74	5005
75	5121
76	5131
77	5246
78	5257
79	5371
80	5494
81	5507
82	5620
83	5872
84	5890
85	5996
86	6009
87	6121
88	6135
89	6245
90	6371
91	6382
92	6495
93	6506
94	6620
95	6631
96	6869
Average	**3323**

response number	**100 register with CPL**
1	11232
2	18532
3	31717
4	44331
5	44583
6	47065
7	47262

8	47381
9	47615
10	49683
11	72783
12	75267
13	77503
14	77595
15	77677
16	81370
17	88972
18	101132
19	101294
20	104384
21	106675
22	107713
23	111792
24	111958
25	123543
26	126773
27	136555
28	139338
29	141996
30	145046
31	152508
32	153302
33	155067
34	158688
35	161950
36	165219
37	176823
38	181043
39	182277
40	184999
41	185084
42	189102
43	205576
44	211259
45	215128
46	217934
47	221426
48	224521
49	228320
50	228320
51	228470
52	238192
53	238318
54	240789
55	244606
56	251482
57	253007
58	261573
59	264080
60	264584
61	264852
62	265109
63	285832

64	290131
65	295430
66	298639
67	298941
68	299156
69	303240
70	308295
71	320048
72	321189
73	323945
74	330597
75	333311
76	340910
77	341585
78	341946
79	347609
80	355499
81	357163
82	366323
83	373348
84	373810
85	378681
86	387458
87	390835
88	391398
89	391583
90	392158
91	397051
92	401663
93	407984
94	408465
95	411227
96	412425
	221773

COS/CCS 2006-3

www.kth.se