UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MARCOS RATES CRIPPA

# Design and Implementation of a Broker for a Service-Oriented Context Management and Distribution Architecture

Undergraduate Thesis presented in partial fulfillment of the requirements for the degree of Bachelor in Computer Science

Prof. Dr. Cláudio Fernando Resin Geyer
Advisor

Prof. Dr.-lng. Hans-Dieter Schotten
Coadvisor

Porto Alegre, July 2010

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-reitor: Prof. Rui Vicente Oppermann
Pró-Reitora de Graduação: Profª. Valquíria Linck Bassani
Diretor do Instituto de Informática: Prof. Flávio Rech Wagner
Coordenador do curso: Prof. João Cesar Netto
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

# ACKNOWLEDGEMENTS

I have various persons and institutions to thank for everything I accomplished. They are listed below, in no particular order of importance.

Thanks to my parents, for the unconditional and infinite support in all my endeavors.

Thanks to my advisor Professor Claúdio Fernando Resin Geyer, for all the knowledge and help, both in the P2PSE project and with this work.

Thanks to Christian Mannweiler, my German coworker, for all the help in implementing this project, and also reviewing this work.

Thanks to my co-advisor, Professor Hans-Dieter Schotten, for giving me the oportunity of going to Germany and participating in this project.

Thanks to the Technische Universität Kaiserslautern, for the excellency and for the great exchange program I participated. I would like to emphasize the great work of the International School for Graduate Students, which help a lot making my stay as easy and smooth as possible.

Thanks to UFRGS and the Informatics Institute, for being one of the best universities and institutes in Brazil, with international recognition. I couldn't have asked for a better place have my Bachelor Studies.

Thanks to all my friends and colleagues, for making this time in Bachelor so pleasant.

Thanks to anyone reading, and I hope you appreciate the work presented here. :)

# CONTENTS

# LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| CB | Context Broker |
| CP | Context Provider |
| CC | Context Consumer |
| CH | Context History |
| REST | REpresentational State Transfer |
| XML | eXtended Markup Language |
| SOA | Service-Oriented Architecture |
| UML | Unified Modelling Language |
| HTTP | Hypertext Transfer Protocol |
| JPA | Java Persistence API |
| API | Application Programming Interface |

# LIST OF FIGURES

# ABSTRACT

This work presents the design and implementation of a Broker for a Service-Oriented Architecture. This architecture manages and distributes context information. The goal is to have an implementation that provides all the services required by the architecture where the Broker resides. It will act as a central point for context information in the whole system, going further than the role normally assigned to a Broker in a SOA.

First, the conceptual base for this work will be presented. The Broker operates in a context aware environment, and has to manage context information. A definition of context and context awareness are presented. Besides, the possible architectural styles and ways for context to be modelled and represented are also presented and discussed.

Subsequently, the architecture in which the Broker operates will be describe and detailed. It's from that architecture that all functional requirements for the Broker shall be determined and extracted. It constitutes a key component to the system, and it should fulfill all requirements in an efficient manner.

All the design and functioning will be then presented. The design choices will be discussed and argued for. Finally, possible expansions and improvements will be discussed.

After the end of this work, it is possible to conclude that the proposed component fulfills all functional requirements. It is a solution that provides a high level context manager, and in no way binded to a specific set of applications. Any client of the system can easily search and access contextual information.

**Keywords:** Context, Context Aware Computing, SOA, Context Aware Systems.

**Design e Implementação de um Broker para uma Arquitetura de Gerência e Distribuição de Contexto Orientada a Serviços**

# RESUMO

Esse trabalho apresenta o design e implementação de um Broker para uma Arquitetura Orientada a Serviços. Essa arqwuitetura gerencia e distribui informação de contexto. O objetivo final é ter uma implementação que provê todos os serviços requiridos pela arquitetura onde esse Broker opera. Ele será um ponto central para informação de contexto no sistema como um todo, tendo um papel maior do que um Broker comum em uma Arquitetura Orientada a Serviços.

Inicialmente, uma base conceitual para esse trabalho será apresentada. O Broker opera em um ambiente consciente de contexto. Uma definição de contexto e de consciência de contexto são apresentadas. Além disso, os possíveis estilos de arquitetura e modos como contexto pode ser modelado e representado também serão apresentados e discutidos.

Subsequentemente, a arquitetura onde o Broker opera será descrita e detalhada. Será a partir dessa arquitetura que os requerimentos funcionais do Broker serão determinados e extraídos. Ele é um componente chave da arquitetura, e deve preencher todos os requerimentos eficientemente.

Todo o design e funcionamento serão então apresentados. As escolhas pertinentes ao design serão discutidas e argumentadas. Finalmente, possíveis expansões e melhorias serão discutidas.

Após o fim deste trabalho, é possível concluir que o componente proposto satisfaz todos os requisitos funcionais. É uma solução que provê um gerente de contexto de alto nível, não estando atrelada a nenhum grupo específico de aplicações. Qualquer cliente do sistema pode procurar e acessar informação contextual facilmente.

**Palavras-chave:** Contexto, Computação Consciente de Contexto, Arquiteturas Orientadas a Serviços, Sistemas Conscientes de Contexto.

# 1 INTRODUCTION

This chapter provides an introduction for this thesis. The subject of this work is the design and implementation of a Context Broker. It operates within an architecture for management and distribution of context information. The motivation behind the work will be discussed. The objectives will be listed and defined. Finally, the organization of the text will be detailed.

## 1.1 Motivation

Every time a user (either a human or a computing system) interacts with a computing system, he's manipulating some form of information. He could be entering his personal data in a social network website, he could be running a bash script in a cluster. Regardless of what he's doing, he's somehow manipulating data to achieve a particular goal.

Sometimes, however, the information the user needs is the information about the environment he's in, or the object he's manipulating is in. So not only does he need to know the name of a mobile phone user, for instance, but also his location, the weather where he is, maybe even his music preferences. All those things form a context where that user is in.

Computers in general, and the Internet specifically, have expanded the presence of computing in the everyday life. One can be 24/7 online with a mobile phone, posting personal data in social networks, checking the timetable for the bus and buying a birthday gift for a friend. All those things can generate context information of some sort, that can be relevant to any user, anywhere.

A challenge then is how to manage all the context information generated by all those different entities along the time. The data must be easily and quickly accessable, regardless of how many are interested in it. It must also be the most up-to-date data possible. Obviously, everyone interested in the data must received it. The connection between the ones consuming the data and the ones generating them must be well estabilished.

Context, context management and context-aware systems are a relatively new and exciting field of research in computing. The research is still trying to find the best approach for things. This work focus in an important component of one of the attempts to solve the management and distribution problem for context information.

This project was developed in the Technische Universität Kaiserslautern, in the research group for Wireless Communications and Navigation of Professor Dr-Ing. Hans-Dieter Schotten. The group is participating in a bigger project called C-CAST [C-CAST. 2010]. The goal was to develop in the group an environment to support research in context and context-awareness, connected to the C-CAST project, but not limited by it. This work details the Context Broker, but the group also works in the Context Providers and in the

Context Consumers.

The project Context Casting (C-CAST) is a collaborative work of many companies, research centers and universities, and its main objective is to evolve mobile multimedia multicasting to exploit the increasing integration of mobile devices with our everyday physical world and environment. It will address three key issues:

- Development of context and group management service enablers for context representation, context assisted group management and context reasoning;
- Definition of a framework to collect sensor data, distribute context information and manage efficiently context aware multiparty and multicast transport;
- Development of mechanisms for autonomous context driven content creation, adaptation and media delivery.

The work presented here falls precisely in the second issue mentioned above: distribution and efficiente management of context information.

## 1.2 Objectives

The main goal of this work is to design and implement a Context Broker for a context management and distribution architecture. The concept of a broker entails that the architecture is service-oriented, and the broker plays a vital role in connecting consumers to service providers. The final product must not only meet all the requirements, but must be stable and easily interface with all other components of the architecture.

The specific objectives are:

- Review the current research in context and context awareness, and define all the basic concepts necessary;
- Detail the architectural setting for this work, with all its components and their behavior;
- Design a broker for that architecture, guaranteeing that it fulfills all the fucntional requirements;
- Implement such broker, following the created design;
- Test the prototype, to ensure correcteness in design and stability in functioning;
- Analyse the results and propose improvements.

## 1.3 Organization of the Text

The remaining text is organized as follows: Chapter 2 discusses the state of the art of the research in context and context management, and also defines the basic concepts used in this work. Chapter 3 details the architecture the Context Broker will operate in, describing all its components and how it models context information. Chapter 4 consists of presenting the requirements the broker must follow, and the final design of the program. Chapter 5 discusses the prototype developed, the development environment, all its modules and all the tests performed. Finally, Chapter 6 concludes this work and discusses possible future work.

# 2 THE CONCEPTS OF CONTEXT AND CONTEXT AWARENESS

This thesis deals with the design and implementation of a Context Broker, that belongs to a Service-Oriented Context Management and Distribution Architecture. It is crucial then to properly define what context is, what context aware computing is, what context management is, and, finally, how context can be modelled and represented.

## 2.1 Context

Although human comunications are heavily based on context, and most people have some understanding on what context is, it is still necessary to provide a simple, yet objective, definition to be used in computing systems. Many definitions have been proposed in the ubiquitous and mobile computing literature.

The first work to center around context awareness, [Schilit. 1994], defines context as location, identities of nearby people and objects, and changes to those objects. A similar definition is presented in [Brown. 1997], where context comprises location, identities of the people around the user, the time of day, season, temperature, etc. [Ryan. 1997] defines context as the user's location, environment, identity and time. [Franklin. 1998] defines it as the situation of the user. [Hull. 1997]'s definition of context is "aspects of the situation". [Dey. 1999] defines context to be the user's physical, social, emotional or informational state. Focusing in the application, [Ward. 1997] and [Rodden. 1998] define context, respectively, as the state of the application's surroundings and as the application's setting. Finally, [Pascoe. 1998] defines context to be the subset of physical and conceptual states of interest to a particular entity.

For this work, the definition provided in [Dey. 2000] will be used. It is a refinement of the definition given by Schilit et al. Context was defined in the following way:

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

This definition is more general than the previous examples, and deals with entities that are relevant to the interaction between application and user. Context is not binded to either the user or the application. This approach allows designers to capture all the elements of the context that are relevant to a design. This definition allows context to be either explicitly or implicitly indicated by the user or application, and also supports a large variety of multimodal contextual information, such as network context, primitive physical context, user/group activities and situations, user/group goals, content meta data, and device capabilities.

## 2.2 Context Aware Systems

Following the definition of context, [Dey. 2000] also provides a definition for context aware systems:

> *A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.*

As with the definition of context, this definition is more general, giving more flexibility for designers, while still encompassing all existing applications.

[Dey. 2000] indicates three categories of features that a context-aware application can support:

- presentation of information and services to a user;
- automatic execution of a service for a user;
- tagging of context to information to support later retrieval.

## 2.3 Context Management

Context Management consists in all the functions required for a context aware system to deal with context. Those functions are:

- Obtaining the context data from different sources;
- Merging correlated context data;
- Locating and accessing context sources;
- Propagating of the acquired data.

The mechanism for context acquisition is important, because it influences to some extent the architectural style chosen for the design of a context aware system. [Chen. 2004] presents three following different approaches on how to acquire contextual information:

- **Direct sensor access**: the context data is directly accessed from the sensors by the clients, with no intermediate layer. This makes the client software and the sensors tightly coupled. This approach does not allow for reusability of sensors, therefore is rarely used;
- **Middleware infrastructure:** uses encapsulation and a layered architecture to hide the low-level sensing details. This allows for great extensibility and reusability;
- **Context server**: extends the middleware based architecture by gathering sensor data in a context server, to facilitate concurrent access. Besides providing great reusability of sensors (like the middleware approach), the usage of a context server also relieves clients of resource intensive operations. The end devices used in context aware systems are usually mobile gadgets with limited processing, storage and power capabilities.

[Winograd. 2001] describes three basic context management models:

- Widgets are derived from GUI elements and provide a public interface for a hardware sensor [Dey. 2001]. They hide low-level sensing details and allow for a reusable and easy application development. They are often controlled by a widget manager. Though increasing efficiency, they are not robust to component failures;

- The networked service model is more flexible and robust, but less efficient compared to the widget approach, because some form of service discovery mechanism must be used;

- The blackboard model represents a data-centric view. Shared media is posted to the blackboard, using asynchronous communication, and is forwarded based on event subscription. Addition of new context sources is simplified whereas communication efficiency decreases.

The broker-based architecture presented in this work (described in the next chapter) uses the context server context acquisition method, and folows the networked service model for context management.

## 2.4 Context Modelling and Representation

Context information needs to be represented and modelled, so it is machine interpretable and exchangeable using well-defined interfaces. This modelling and representation needs to support easy manipulation and extension goals, efficient search and scalability.

[Strang. 2004] determines the generic requirements for any modelling approach: to be able to cope with high dynamics and distributed processing and composition; allow for partial validation independently of complex interrelationships; enable rich expressiveness and formalism for a shared understanding; indicate richness and quality of information; must not assume completeness and unambiguousness; be applicable to existing infrastructures and frameworks.

Context models can be classified into six major categories [Strang. 2004]:

- Key-value pairs form a simple tuple of information. The context information is assigned to a unique key in order to allow for easy lookup by applying a matching algorithm. These pairs are easy to manage but lack structure and therefore do not allow for efficient context retrieval;

- Markup scheme models incorporate a hierarchical data structure of markup tags, attributes and content. Well known examples are the User Agent Profile and the Composite Capabilities/Preference Profile [CCPP. 2010] being based on XML;

- Graphical models (e.g. based on UML) allow for a picturesque description of a context model [Sheng. 2005] and for deriving an Entity Relationship model as required in rational databases. An extension is proposed by [Henricksen et al. 2005], introducing the Object-Role Modelling (ORM);

- Object oriented models offer powerful capabilities of inheritance, reusability and encapsulation. Access of contextual information is provided by well defined interfaces [Hofer et al. 2003];

- Logic based models offer a high degree of formalism and typically comprise facts, expressions and rules. They enable formal inference by means of general probability logic, description logic, functional logic or first-order predicate logic;
- Ontological modelling intends to capture an abstract conceptual vision of the world. Ontology is commonly described by using languages standardised by the World Wide Web Consortium (W3C) in the context of the semantic web. Most relevant are the Resource Description Framework Schema (RDF-S) and the Web Ontology Language (OWL).

Whatever model is chosen, meta-data should be inserted. That meta-data should specify attributes as: origin of the data, time of creation or detection, to which entity it relates, and so on. In that way, data can be properly differentiated and handled.

Researchers in [Strang. 2004] [Baldauf. 2007] [Wang et al. 2004] concluded that ontologies are in principle the best way to represent and model context due to the extendibility and unambiguousness.

The model chosen for the broker-based architecture this work deals with was the markup model. The network nature of the messages (HTTP messages, in this case) facilitates textual, non graphic model. There is no need for the insertion of another layer of abstraction. This is the common approach used in SOAs. An XML-based language was created, called ContextML. It is described in the next chapter.

# 3 DESCRIPTION OF THE CONTEXT MODEL AND ARCHITECTURE

The Context Broker described in this document exists within an architecture. It is necessary then to fully describe this architecture, so that the role played by the Broker can be understood. The description given here focuses on the modeling of the context information and the components of the architecture.

Initially, the basic definitions of Context Entity and Context Scope (from now on, referred just as entity and scope) are explained. They are fundamental to understanding the way that context information was model in the architecture.

The three basic components of the architecture are the Context Consumer (CC), the Context Broker (CB) and the Context Provider (CP). Provider, Broker and Consumer are the three basic components of Service-Oriented Architectures (SOAs). The Provider allows for the desired data and functions to be accessed. The Consumer is the process looking for that data and functions. The Broker is the general index, that connects the consumers to the providers. Since this is a Context Management and Distribution SOA, all the components deal with context information.

The description of the responsibilities and interfaces for these components will be presented in this chapter.

## 3.1 Basic Definitions

### 3.1.1 Context Entity

An entity is the subject which context data refers to, and it is composed by a type and an identifier. Every exchange of context data is associated with a specific entity, which can be a complex group of more than one entity. Every CP supports one or more entity types.

The entity identifier specifies a particular item in a set of entities belonging to the same type. A type is an object that categorizes a set of entities. Examples of entity types are:

- **username** : for human users;
- **imei** : for mobile devices;
- **mobile** : GSM phone number for mobile users;
- **SIP uri** : for SIP accounts;
- **groupid** : for groups of other entities.

Consider the example of a CP that provides geographical cell-based location for mobile devices. If the location information is obtained from the computation of parameters provided by mobile devices, this CP supports entity type *mobile*. When the CB receives a request for the location of a human user (entity type username), the broker is not able to invoke the provider previously described because it does not support this entity type, but if the user has a mobile device, information about his location is probably available in the system.

### 3.1.2 Context Scope

A set of closely related context parameters is defined as "scope" . Every context parameter has a name and belongs to only one scope. Parameters in a scope are always requested, updated, provided and stored at the same time. Data creation and update within a scope are always atomic and context data in a scope is always consistent. Scopes themselves can be atomic or aggregated, as a union of different atomic context scopes.

For example, consider the scope **position**, which refers to the geographic position of an entity. This scope could be composed of parameters *latitude*, *longitude* and *accuracy* (margin of error) and these are always changed at the same time. Updating the latitude value without updating longitude, if it is changed, and vice versa, is clearly not correct.

The association between entity and scope is presented in Figure 3.1:

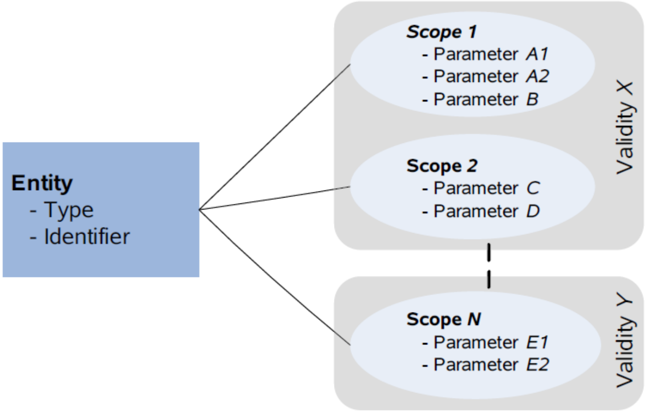


Figure 3.1: Entity-Scope Relationship

Every scope that is exchanged is tagged with its timestamp and expiration time. The expiration time tag states the validity of the scope. After this time, the information is considered to be invalid, and should be deleted. The Context Provider that generated the context data is in charge of setting the expiration time.

## 3.2 Context Modeling and Markup Language

Context modeling refers to the process of managing information related to context provided by different entities in a system and is a vital process in a context aware system. The context information needs to be represented and modeled for being machine interpretable and exchangeable using well-defined interfaces. The goals are to support easy manipulation (low overhead in keeping the model up-to-date), easy extension (simple mechanism for adding new types of information), efficient search and query access and scalability.

ContextML is an XML-based language designed for use in the architecture as a common language for exchanging context data between architecture components. It defines a language for context representation and communication between components that should be supported by all components in the architecture.

ContextML allows the following features:

- Representation of context data;
- Announcement of CPs towards the CB;
- Description of CPs published to the CB;
- Description of context scopes available on CB;
- Representation of generic response (ACK/NACK).

In this section, a general overview of the ContextML will be given. The full XML Schema is presented in the Appendix.

The ContextML schema is composed by:

- **ctxEls**: contains one or more context elements;
- **ctxAdvs**: contains the announcement of CPs;
- **scopeEls**: contains information about scopes available to the CB;
- **ctxPrvEls**: contains information about CPs published to the Context Broker;
- **ctxResp**: contains a generic response from a component.

### 3.2.1 Announcement of a Context Provider

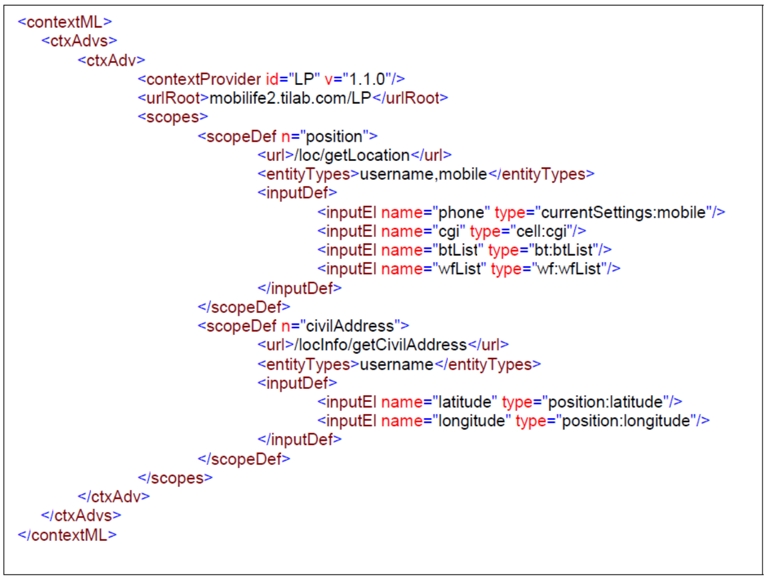
A Context Provider has to publish its presence to the CB. When a CP starts, it sends to the CB a ContextML message in which it specifies its name, its version, the entity types and the context scopes that are supported. Moreover the CP has to publish the URL to invoke and the input parameters it needs for context computation (if necessary). In this way, when the CB receives a request for context information of a specific entity and a specific scope, it knows which CPs can provide that information. Figure 3.2 shows an example of Context Provider announcement.

### 3.2.2 Description of Context Providers

ContextML allows the representation of a list of Context Providers. This list is given when the CB receives a request for which provider can give context information related to a specific entity and scope. Figure 3.3 shows an example of a description of available Context Providers.

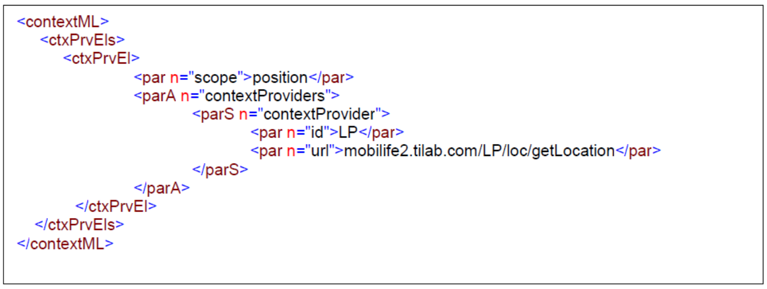
### 3.2.3 Description of Context Scopes

The ContextML language allows the representation of the list of scopes. Figure 3.4 shows an example of a description of context scopes.



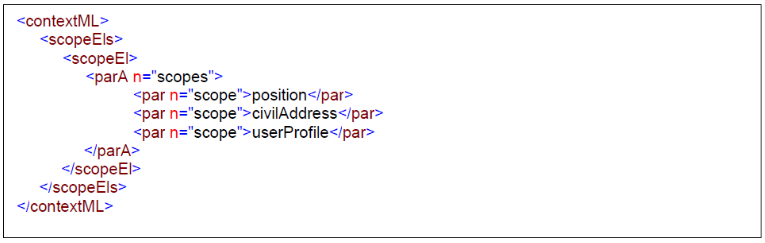
```
<contextML>
   <ctxAdvs>
      <ctxAdv>
         <contextProvider id="LP" v="1.1.0"/>
         <urlRoot>mobilife2.tilab.com/LP</urlRoot>
         <scopes>
            <scopeDef n="position">
               <url>/loc/getLocation</url>
               <entityTypes>username,mobile</entityTypes>
               <inputDef>
                  <inputEl name="phone" type="currentSettings:mobile"/>
                  <inputEl name="cgi" type="cell:cgi"/>
                  <inputEl name="btList" type="bt:btList"/>
                  <inputEl name="wfList" type="wf:wfList"/>
               </inputDef>
            </scopeDef>
            <scopeDef n="civilAddress">
               <url>/locInfo/getCivilAddress</url>
               <entityTypes>username</entityTypes>
               <inputDef>
                  <inputEl name="latitude" type="position:latitude"/>
                  <inputEl name="longitude" type="position:longitude"/>
               </inputDef>
            </scopeDef>
         </scopes>
      </ctxAdv>
   </ctxAdvs>
</contextML>
```

Figure 3.2: Context Provider Advertisement



```
<contextML>
   <ctxPrvEls>
      <ctxPrvEl>
         <par n="scope">position</par>
         <parA n="contextProviders">
            <parS n="contextProvider">
               <par n="id">LP</par>
               <par n="url">mobilife2.tilab.com/LP/loc/getLocation</par>
            </parS>
         </parA>
      </ctxPrvEl>
   </ctxPrvEls>
</contextML>
```

Figure 3.3: List of Context Providers



```
<contextML>
   <scopeEls>
      <scopeEl>
         <parA n="scopes">
            <par n="scope">position</par>
            <par n="scope">civilAddress</par>
            <par n="scope">userProfile</par>
         </parA>
      </scopeEl>
   </scopeEls>
</contextML>
```

Figure 3.4: List of Scopes

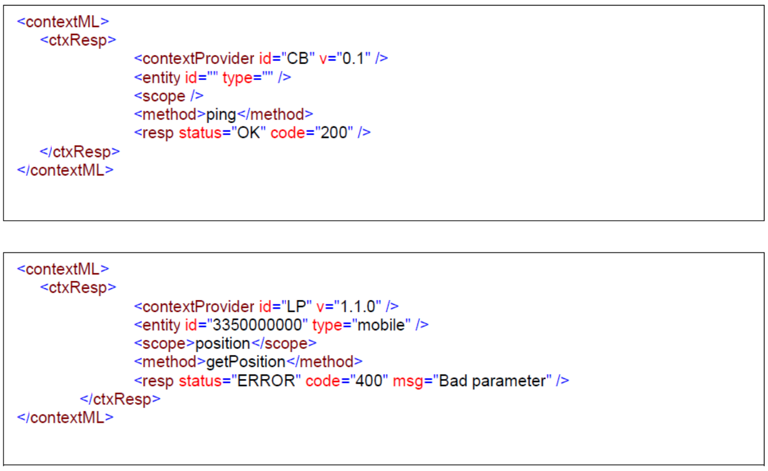
### 3.2.4 Generic response

ContextML allows the representation of generic responses intended as acknowledgement (ACK) or non-acknoledgement (NACK) between platform modules. Figure 3.5 shows examples of ACK and NACK messages.

### 3.2.5 Context Data

Any context information given by a provider refers to an entity and a specific scope. When a context provider is queried, it replies with a ContextML message which contains the following elements:

- **ContextProvider**: a unique identifier for the provider;



```
<contextML>
    <ctxResp>
                    <contextProvider id="CB" v="0.1" />
                    <entity id="" type="" />
                    <scope />
                    <method>ping</method>
                    <resp status="OK" code="200" />
    </ctxResp>
</contextML>
```

```
<contextML>
    <ctxResp>
                    <contextProvider id="LP" v="1.1.0" />
                    <entity id="3350000000" type="mobile" />
                    <scope>position</scope>
                    <method>getPosition</method>
                    <resp status="ERROR" code="400" msg="Bad parameter" />
          </ctxResp>
</contextML>
```

Figure 3.5: ACK and NACK messages

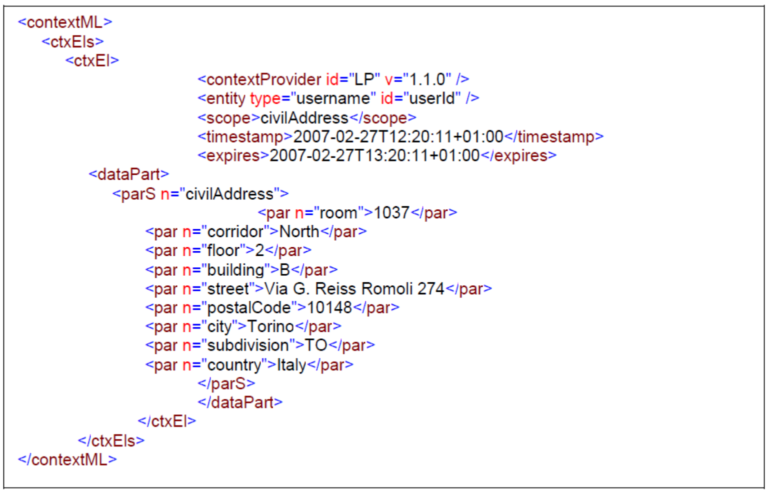
- **Entity**: the identifier and the type of the entity to which the data is related to;

- **Scope**: the scope to which the context data belongs to;

- **Timestamp and Expires**: respectively, the time in which the data was created, and the expiration time of the data part;

- **DataPart**: part of the message which contains the actual context data which is represented by a list of a features and relative values through the `<par>` element (parameter). The data can be grouped through the `<parS>` element (parameter structure) and/or `<parA>` element (parameter array) if necessary.

Figure 3.6 shows examples of a Context Elements message.



```
<contextML>
    <ctxEls>
        <ctxEl>
                              <contextProvider id="LP" v="1.1.0" />
                              <entity type="username" id="userId" />
                              <scope>civilAddress</scope>
                              <timestamp>2007-02-27T12:20:11+01:00</timestamp>
                              <expires>2007-02-27T13:20:11+01:00</expires>
            <dataPart>
                <parS n="civilAddress">
                                         <par n="room">1037</par>
                      <par n="corridor">North</par>
                      <par n="floor">2</par>
                      <par n="building">B</par>
                      <par n="street">Via G. Reiss Romoli 274</par>
                      <par n="postalCode">10148</par>
                      <par n="city">Torino</par>
                      <par n="subdivision">TO</par>
                      <par n="country">Italy</par>
                              </parS>
                              </dataPart>
                    </ctxEl>
          </ctxEls>
</contextML>
```

Figure 3.6: Context Elements message

## 3.3 Context Consumer

### 3.3.1 Description

A Context Consumer is an application that uses context data. A CC can retrieve context information in various ways:

- **Direct request to the CB :** in that case, the CB tries to locate the context data requested, either in its cache, in the Context History or contacting a CP;
- **Direct request to a CP :** the CC sends a request to a particular CP;
- **Lookup Service :** the CC asks the Lookup service of the CB to provide the available CPs for a certain entity and scope, and then sends a direct request to those CPs;
- **Subscription Service :** subscribing to context information in the CB;
- **Searching the Context History :** if the CC wants historical data, then it can search the Context History.

### 3.3.2 Responsibilities

- **Discover CB :** Discover the CB in the network. Requires the CB to be discoverable through some protocol and requires CCs to use the discovery mechanism of that protocol;
- **Find CPs :** Find CPs of interest by asking the CB. The CB provides a lookup mechanism and the CCs use that lookup mechanism, by providing the entity and scope they are interesting in;
- **Query CPs for context :** Query CPs for context information. Requires CPs to advertise its context provision features and input parameters;
- **Subscribe to the CB :** Subscribe to the CB for context information. Requires the CB to provide an asynchronous subscription service. Moreover, requires the CB to publish context information received from the CPs.

## 3.4 Context Provider

### 3.4.1 Description

A Context Provider (CP) is a component whose task is to provide context information. Context Provider will achieve this by gathering data from a collection of sensors, network, services (e.g. web services) or other relevant sources. The CP will use various filtering, aggregation and reasoning mechanisms to infer context from raw sensor, network or other source data. This created context will be modeled according to the ContextML. Each provider will be tailored to provide a particular type of context, e.g. Calendar Provider will provide calendar information about an entity, Location Provider will provide location information about an entity. Every CP registers its availability and capabilities by sending appropriate announcements to the CB and exposes interfaces to provide context information to the CB and to Context Consumers.

CPs have three methods for providing information:

- **Direct Request :** Every CP has an interface for a direct request for context data. The format of the interface (URL, parameters necessary) is describe in the advertisement given to the CB;

- **Context Update :** All the registered CPs are required to send context data to the CB with a certain frequency. This is required so that the data in the Context Cache in the CB is up-to-date, and can be accessed by the CCs. Moreover, by sending data regularly, the CP can signal to the CB that it is still operating properly;

- **Subscription :** A CC can subscribe to some entity and scope in the CB. When any CP sends a context update that relates to that subscription, the CB will forward the data to the CC. In this way, context data can be asynchronously provided by the CP.

### 3.4.2 Responsibilities

- **Discover CB :** Discover the CB in the network. Requires the CB to be discoverable through some protocol and requires CPs to use the discovery mechanism of that protocol;

- **Advertise Features and Register in the CB :** Inform CB of its existence, direct request interfaces and acceptable input parameters;

- **Provide Context Information :** Accept context queries on registered interface. The CP then processes the query so that it can reply with the most suitable context information. Finally, delivers context information requested through a query to the requesting entity (usually the CB);

- **Data Acquisition, Filtering, Aggregation, and Reasoning :** The basic internal task of the CP is creating context. It communicates with sensors, networks, and/or other services to gather data. Then it processes internally and models according to the ContextML, in order to provide them to the system.

## 3.5 Context Broker

### 3.5.1 Description

The Context Broker is the key component of the architecture. It works as a handler of context data and as an interface between other architectural components. Primarily the CB has to control the context flow among all attached components. In order to do that, the CB has to know every CP; has to be a centralized point for up-to-date data; has to push new context data to all interested consumers; and has to keep a historical archive of all received data. All those functions are implemented either internally by the Broker (Register Table, Publish-Subscribe System and Context Cache), or implemented outside the Broker (Context History), but connected to it. The inner workings of the Broker will be explored in the next chapter.

### 3.5.2 Responsibilities

- **Discoverable :** The CB must be discoverable in the network. Requires some type of protocol and requires CPs and CCs to use the discovery mechanism of that protocol;

- **Context Provider Registration :** CB needs to provide a registration mechanism for the CPs and to keep a record of all known and functioning CPs in a table. Requires a registration interface that can be used by CPs after discovering the Broker;

- **Context Provider Lookup :** CCs will ask the CB for CPs of interest. It is the job of the CB to provide the CCs with a list of CPs based on the criteria given by the CC. Requires a lookup mechanism that can return matching CPs from the CP record based on CCs requested criteria;

- **Context Subscription :** CCs may need context in asynchronous mode. The CB, in this case, has to provide a subscription service so that the CCs can subscribe with CB for a particular entity and scope. Requires definition of the subscription format and interface;

- **Context Notification :** Based on the matching done by CPs (that was based on CC subscriptions forwarded to CPs by the CB), the CB is responsible for forwarding the matched context to the subscribed CCs. Requires an asynchronous notification mechanism served by the CB to the CCs;

- **Context Cache :** For performance reasons, the Broker is required to keep all up-to-date context data in a cache. The data is regularly sent by the Context Providers. If a CC wants the latest data related to a specific entity and/or scope, it can directly ask the Broker. Requires the Broker to have an internal table to hold the data, and a mechanism for the providers to send their data;

- **Context History :** The Broker should not discard outdated data. Any statistical or temporal analysis that CCs or even CPs need to do will require a historical archive. Although this service is external to the Broker, it is responsible for sending the expired or replaced data in the Context Cache to the Context History. Requires the CB to have a special interface with the History, specifically designed to add data to the archive.

## 3.6 Communications between Components

Figure 3.7 shows all the interactions presented in the above described architecture.

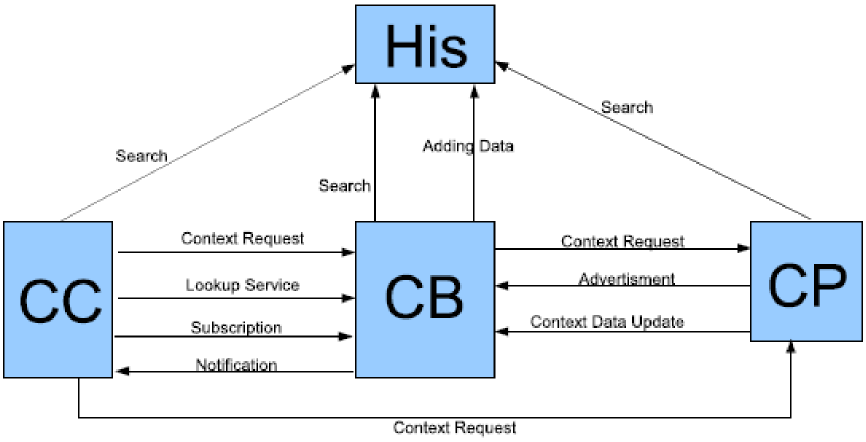


Figure 3.7: Comunications between all the Components

Not all messages are shown in the picture: ACKs and NACKs were left out, also the response from Context Request. Those messages are assumed implicit and removed for clarity reasons.

It is important to emphasized that the roles of CP and CC are not fixed. There is nothing in the architecture that forbids a CP to make a Context Request to the Broker, thus playing the role of a consumer. Those distinctions are not made by the CB.

# 4 DESIGN OF THE CONTEXT BROKER

The design and implementation of the Context Broker will be discussed in this chapter. In order to properly describe all the components, a modelling language that provides all the tools and diagrams is necessary. This work uses the Unified Modelling Language[UML. 2010]. UML has the tools to specify, visualize, modify, construct and document all the artifacts of an application.

## 4.1 Requirements

First in the development cycle, one should identify all the requirements of the application. Only when they are defined it's possible to determined the steps each component has to take to fulfill them.

### 4.1.1 Hierarchical Diagram of Functions

Before starting the use of the UML, a concise and simple list of the requirements should be made. This list is going to facilitate the creation of the Use Cases (presented in the next subsection). All Use Cases will address one or more of the functions listed below.

1. **Register of CPs**
   (a) Receive CP Advertisement
   (b) Check CP Status
2. **Lookup Service**
   (a) Receive Lookup Request
   (b) Answer to Lookup Request
3. **Cache Use**
   (a) Add Data to The Cache
   (b) Check Validity of Context Data
   (c) Search for Context Data
4. **Publish/Subscribe**
   (a) Subscription Request from CC
   (b) Check if Received Data is Subscribed

    (c) Check if an Subscription is Expired

5. **Database Use**
    (a) Add Data to the History
    (b) Search the Database

6. **Context Query**
    (a) Receives Data from CP
    (b) Receive Context Data Request

The lookup and registering services constitute the basic duties of a broker in a SOA. With them, the Broker has the awareness of the existence and status of the CPs.

All the other functions are related with storing and providing context data. Publish/Subscribe deals with asynchrounous data providing, through a subscription system. Database Use is implemented through the Context History, and deals with out-of-date data. It's important to point out that only the Broker is qualify to send data to the History, being the only component keeping an updated storage of all the context data in the system (the Cache). Cache Use lists the basic functions of this up-to-date context data storage. All context data has a validity attribute attached to it. After a certain given time, the data should be transfer to the History and remove from the Cache. Context Query represents the interaction with the Context Consumer. For the CC, the Broker is a centralized point from which up-to-date data can be retrieve.

### 4.1.2 Use Cases

A use case describes the behavior of the system, given a request from outside the system. In this project, there are four actors: Context Broker, Context Provider, Context Consumer and Context History. The following list details all the use cases present in the system.

#### *4.1.2.1 Register of CPs*

- **Name :** Receive CP Advertisement
- **Actor(s) :** Context Broker, Context Provider
- **Objective :** Update the registry information related to a CP, or receive information about a new CP to be inserted in the Register Table
- **Description :** When a CP starts its operation, it should send an advertisement to the CB, to announce its existence and capabilities. If the data provided changes somehow, the CP should also send another advertisement, updating its registry in the CB. This assumes that the CB will be already operating
- **Type :** Primary and Essential
- **References :** 1.a, 1.b
- **Sequence of Events :**
    1. The CP sends a context advertisement message to the CB

2. The CB analizes the message
    (a) If the message is not properly formed, the CB sends a NACK message as a reply
    (b) Otherwise, the CB checks if this is a new CP
        i. If it's a new CP, creates a new entry in the Register Table, and inserts the data of the message in it
        ii. If it's an old CP, updating the correspondent entry with the new data
        iii. If any error occurs, a NACK message is sent
3. A timer is created (or resetted, if already existed) and associated with this entry (see 1.b)
4. The CB sends an ACK message

- **Name :** Check CP Status
- **Actor(s) :** Context Broker, Context Provider
- **Objective :** Check if a CP that hasn't sent any message after a certain time is still online or not. If it is offline, then it's mark disconnected in the Register Table
- **Description :** All registered CPs are required to advertise or send new context data with a certain frequency. If a CP fails to do so, the CB will try to contact it, and determine its status. Every entry in the Register Table has a timer associated to it. This timer is responsible for triggering this use case
- **Type :** Primary and Essential
- **References :** 1.b
- **Sequence of Events :**
    1. The timer associated with a entry in the Register Table is triggered
    2. The CB sends a message, trying to access the getContext interface of the CP
        (a) If no response is given, the entry is marked as disconnected
        (b) Otherwise, the timer is reset

#### *4.1.2.2 Lookup Service*

- **Name :** Receive Lookup Request
- **Actor(s) :** Context Broker, Context Consumer
- **Objective :** Provide a CC with a list of registered CPs that fits the searching criteria (scope and entity type)
- **Description :** The CB is the only point in the system that keeps a record of all the CPs. If a CC needs to know which CPs to contact or subscribe to in order to get data related to a scope and entity type, it should send a lookup request to the CB

- **Type :** Primary and Essential
- **References :** 2.a
- **Sequence of Events :**
  1. The CC accesses the REST interface getContextProviders(entitytype,scope)
  2. The CB analizes the parameters
     (a) At least one scope has to be provided, the entity type is optional. If a scope is not given, the CB sends a NACK message as an answer
     (b) Otherwise, search the Register Table for CPs that handle data for that scope and entity type.
  3. Answer to Lookup Request case begins

- **Name :** Answer to Lookup Request
- **Actor(s) :** Context Broker, Context Consumer
- **Objective :** Send the requesting CC the result of the search in the Register Table
- **Description :** The CB will take the data provided by the Receive Lookup Request case (a list of CPs), put it in a message and send it to the CC.
- **Type :** Primary and Essential
- **References :** 2.b, 2.a
- **Sequence of Events :**
  1. The CB sends a provider lookup response message back to the CC
     (a) If the list is empty, a NACK message is sent

#### *4.1.2.3 Cache Use*

- **Name :** Add Data to The Cache
- **Actor(s) :** Context Broker, Context Provider
- **Objective :** Insert or renew context data present in the Cache
- **Description :** When the CB receives a context update from a CP, it puts it in the Cache. If there is already an entry for that entity and scope, it's sent to the History and renewed by the new data.
- **Type :** Primary and Essential
- **References :** 3.a, 3.b
- **Sequence of Events :**

1. The CP sends a context update message to the CB
2. The CB analizes the message
   (a) If the message is not properly formed, the CB sends a NACK answer message
   (b) Otherwise, the CB checks if this CP exists
      i. If it doesn't, a NACK message is sent
      ii. If the cache entry already exists, Add Data to the History case begins, with the old entry
3. The new entry is inserted in the Cache
4. A timer is created (with the duration time given in the entry's timestamps) and associated with this entry (see 3.b)
5. The CB sends an ACK message

- **Name :** Check Validity of Context Data
- **Actor(s) :** Context Broker
- **Objective :** When an entry of the Cache becomes out-of-date, its data will be stored in the History, and the entry deleted
- **Description :** When the Broker receives a message with context data, the CP needs to indicate in which period of time that data is considered valid. Every entry has a timer associated to it. The timer will go off when the time given by the CP is up.
- **Type :** Primary and Essential
- **References :** 3.b
- **Sequence of Events :**

  1. A timer in an entry reaches it's time
  2. Add Data to the History case begins
  3. The entry associated with that timer is deleted

- **Name :** Search for Context Data
- **Actor(s) :** Context Broker
- **Objective :** When a CC requests context data, the first place the Broker will search is the Cache, where the data is up-to-date.
- **Description :** Given the entity id and type, and a list of scopes, the Cache is searched to find the correspondent entries.
- **Type :** Primary and Essential
- **References :** 3.c, 6.b

- **Sequence of Events :**
  1. The CB searches in the Cache for entries with the entity id and type, and scopes given
  2. It returns the data found to the class responsible with handling the requests for context data

#### *4.1.2.4 Publish/Subscribe*

- **Name :** Subscription Request from CC
- **Actor(s) :** Context Broker, Context Consumer
- **Objective :** A context consumer subscribes to an entity id, entity type and scope.
- **Description :** The Broker provides a subscription system. When the CB receives data with the entity id, entity type and scope subscribed, it forwards the data to the consumer
- **Type :** Primary and Essential
- **References :** 4.a
- **Sequence of Events :**
  1. The CC access the REST interface subscribe(entityID, entity type, scopeList, callbackUrl, time)
  2. The CB analizes the parameters
     (a) If any parameters is not given, or the time given is less than one, the CB sends a NACK message as an answer
     (b) Otherwise, the CB checks if there is a CP that provides the specified data
         i. If there isn't, a NACK message is sent
  3. A new entry is inserted in the Publish/Subscribe table
  4. A timer is created (with the duration time given in the parameters, in minutes) and associated with this entry (see 4.c)
  5. The CB sends an ACK message

- **Name :** Check if Received Data is Subscribed
- **Actor(s) :** Context Broker
- **Objective :** When new context data arrives, the CB checks if there are any subscriptions to it
- **Description :** The subscription system is the way a Consumer signalizes that it wishes to receive data when that data arrives at the Broker. It is totally asynchronous in nature. So, it is the Broker's responsibility to guarantee that all consumers interested in a certain data receive it.

- **Type :** Primary and Essential
- **References :** 4.b
- **Sequence of Events :**
    1. The CB searches through the Publish/Subscribe table for subscriptions to that entity id, entity type and scope
    2. The CB sends the data to every matched subscription, using the callback URL provided in the subscription

- **Name :** Check if an Subscription is Expired
- **Actor(s) :** Context Broker
- **Objective :** The subscription system operates using timers, with every subscription lasting an amount of time determined by the subscriber.
- **Description :** Just like the entries in the Cache, every entry in the Publish/Subscribe Table has an associated timer. When the time runs up, that entry is deleted. It's the responsiblity of the Context Consumer to guarantee that its subscription is always valid in the Broker.
- **Type :** Primary and Essential
- **References :** 4.c
- **Sequence of Events :**
    1. A timer in an entry of the Publish/Subscribe Table reaches it's time
    2. The entry associated with that timer is deleted

#### *4.1.2.5 Database Use*

- **Name :** Add Data to the History
- **Actor(s) :** Context Broker, Context History
- **Objective :** When data gets out-of-date in the Cache, either because new data came to replace it or its validity expired, the Broker will send it to the Context History
- **Description :** The Context History keeps a record of all the data sent to the Broker. So it's the Broker's responsibility to send the data to the History, and guarantee it's out-of-date and complete.
- **Type :** Primary and Essential
- **References :** 5.a, 3.a, 3.b
- **Sequence of Events :**

1. The CB sends a message for context update to the Context History
2. The CH analizes the message
   (a) If the message is not properly formed, the CH replies with a NACK message
   (b) Otherwise, the CH inserts the data in its database
3. The CH sends an ACK message

- **Name :** Search the Database
- **Actor(s) :** Context Consumer, Context History
- **Objective :** The Context History acts as a special CP. Only the CB can insert data, but any CC can access it.
- **Description :** Any consumer that needs historic data will access the Context History to get it. There will be multiple parameters, so the consumer can be as specific as it needs in the search.
- **Type :** Primary and Essential
- **References :** 5.b
- **Sequence of Events :**
  1. The CC accesses the REST interface search(cpID, entityID, entityType, scope, begin, end) (the details about the search will be detailed in a later section)
  2. The CH analizes the parameters
     (a) If no parameters are given, the CH sends a NACK message as answer
     (b) Otherwise, the CH searches its database
         i. If no entry matching the criteria is found, the CH sends a NACK message as answer
  3. The CH sends a message with the result list

#### *4.1.2.6 Context Query*

- **Name :** Receives Data from CP
- **Actor(s) :** Context Broker, Context Provider
- **Objective :** Request to one or more CPs data not found in the Cache
- **Description :** If the CB doesn't have the data readily available in the Cache, it will ask for it from all capable CPs. If the CP is online, it will send a response to that request.
- **Type :** Primary and Essential

- **References :** 6.a, 6.b
- **Sequence of Events :**
  1. The CB receives a context request message
  2. The CB analizes the message
     (a) If the message is not properly formed, the CB discards it
     (b) If the CP can't provide the data, the CB discards the message
     (c) If any data is received, it's returned to the class responsible with handling the requests for context data

- **Name :** Receive Context Data Request
- **Actor(s) :** Context Broker, Context Consumer
- **Objective :** Provide context data to interested Context Consumers
- **Description :** The CB acts as a centralized point in the architecture for up-to-date context data. The data can either come from its own Cache or from a request to a CP. The actual origin is irrelevant to the Consumer, and it can't make that distinction in the request.
- **Type :** Primary and Essential
- **References :** 6.b
- **Sequence of Events :**
  1. The Context Consumer accesses the REST interface getContext(entity,type,scopeList)
  2. The CB analizes the parameters
     (a) At least one scope must be given. If it's not given, or only entity information is provided, the CB sends a NACK message as a reply
     (b) Otherwise, Search for Context Data case Begins
        i. If no Cache entry matching the criteria is found, the CB searches the Register Table for CPs that can provide that data
        ii. The CB sends a direct request for data to each matching CP
        iii. Receive Data from CP case begins
        iv. If no data is returned (either the CPs can't provide it, or they are unavailable), then the CB sends a NACK message as an answer
  3. The CB sends a message with the data found

### 4.1.3 Use Case Diagram

The Use Case Diagram will give a complete picture of all the actors and actions present in the whole system. For the purpose of clarity, an overview of the use case diagram will be shown, then each module (Cache Use, Database Use, etc..) will be separately presented and detailed.

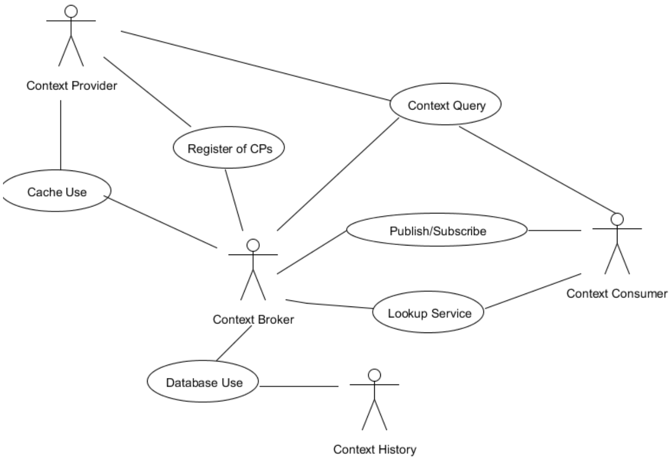


Figure 4.1: The Overview

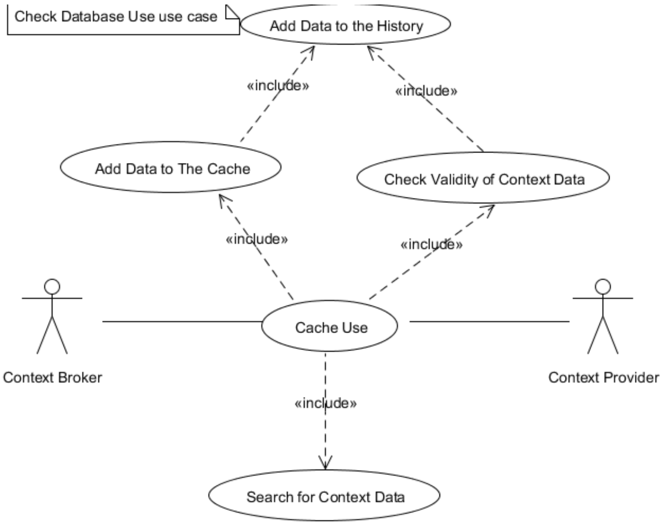


Figure 4.2: Cache Use use case

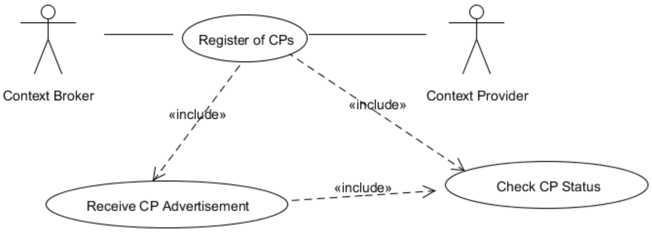


Figure 4.3: Register of CPs use case

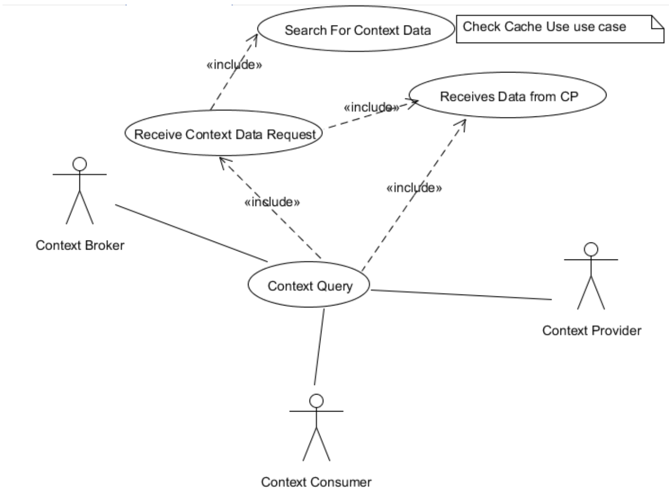


Figure 4.4: Context Query use case

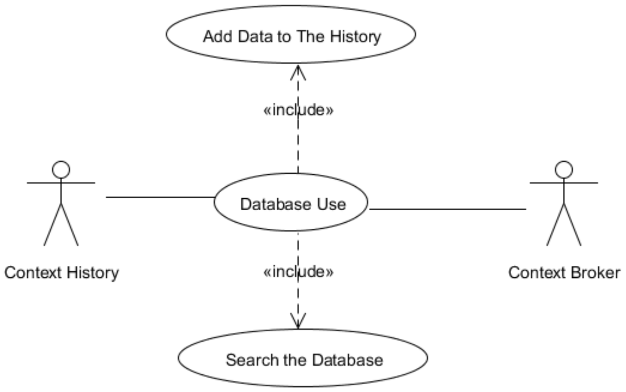


Figure 4.5: Database Use use case

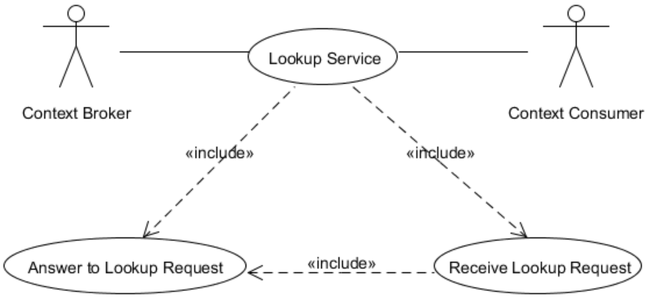


Figure 4.6: Lookup Service use case

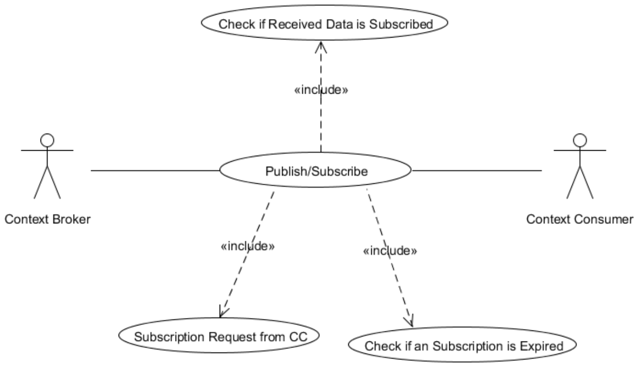

Check if Received Data is Subscribed
«include»
Context Broker
Publish/Subscribe
Context Consumer
«include»
«include»
Subscription Request from CC
Check if an Subscription is Expired

Figure 4.7: Publish/Subscribe use case

## 4.2 Specification

In this stage of the design process, all the steps that must be taken by each actor to execute all the use cases are determined. UML has an artifact called Message Sequence Charts that does exactly that, showing not only all the interactions, but also the order in which they occur.

### 4.2.1 Message Sequence Charts

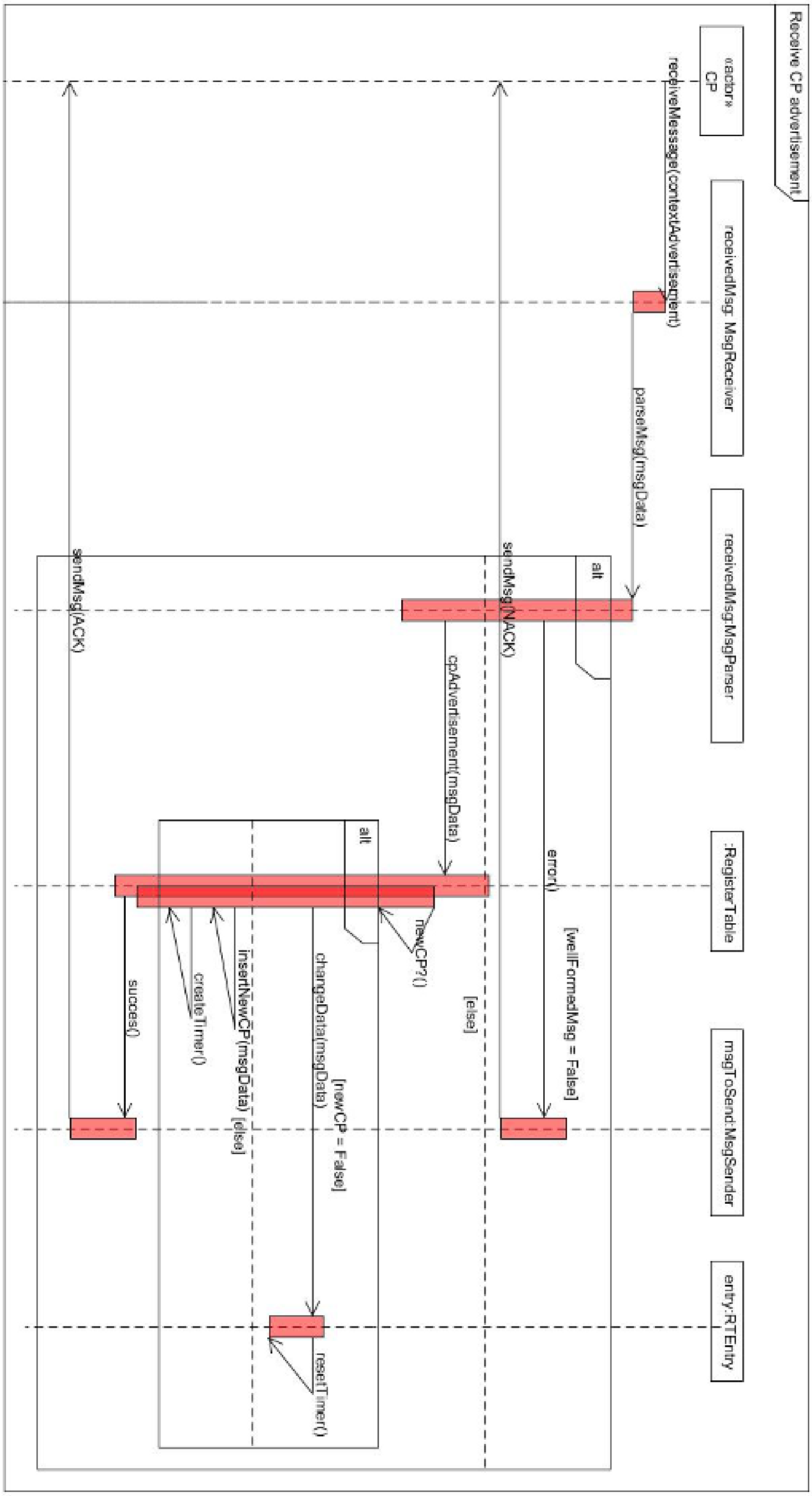


Figure 4.8: Receive CP Advertisement

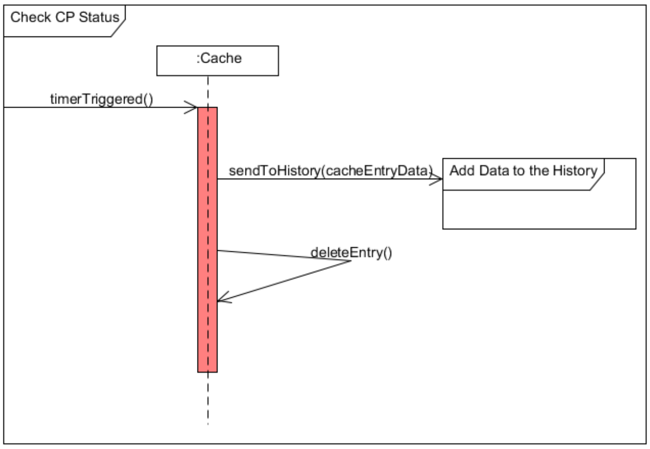


Figure 4.9: Check CP Status

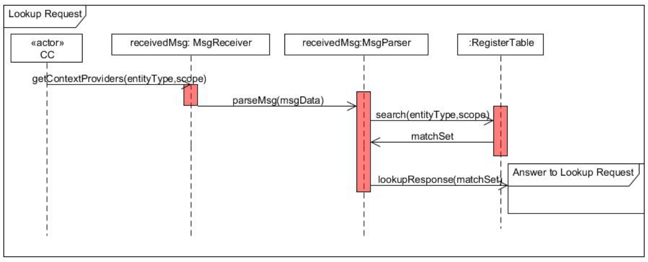


Figure 4.10: Lookup Request

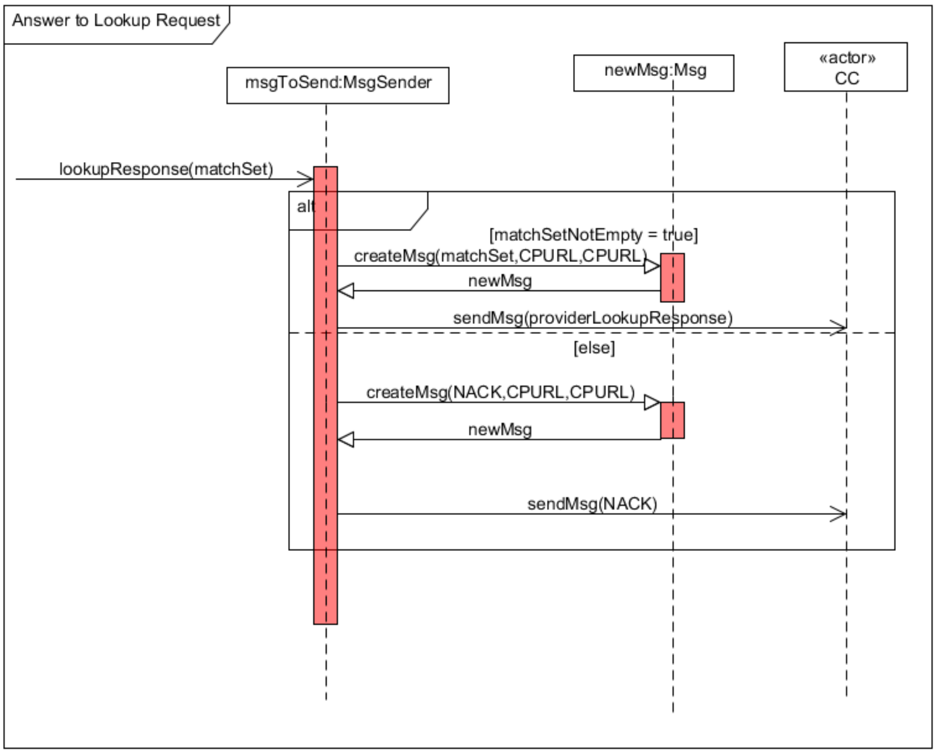


Figure 4.11: Answer To Lookup Request

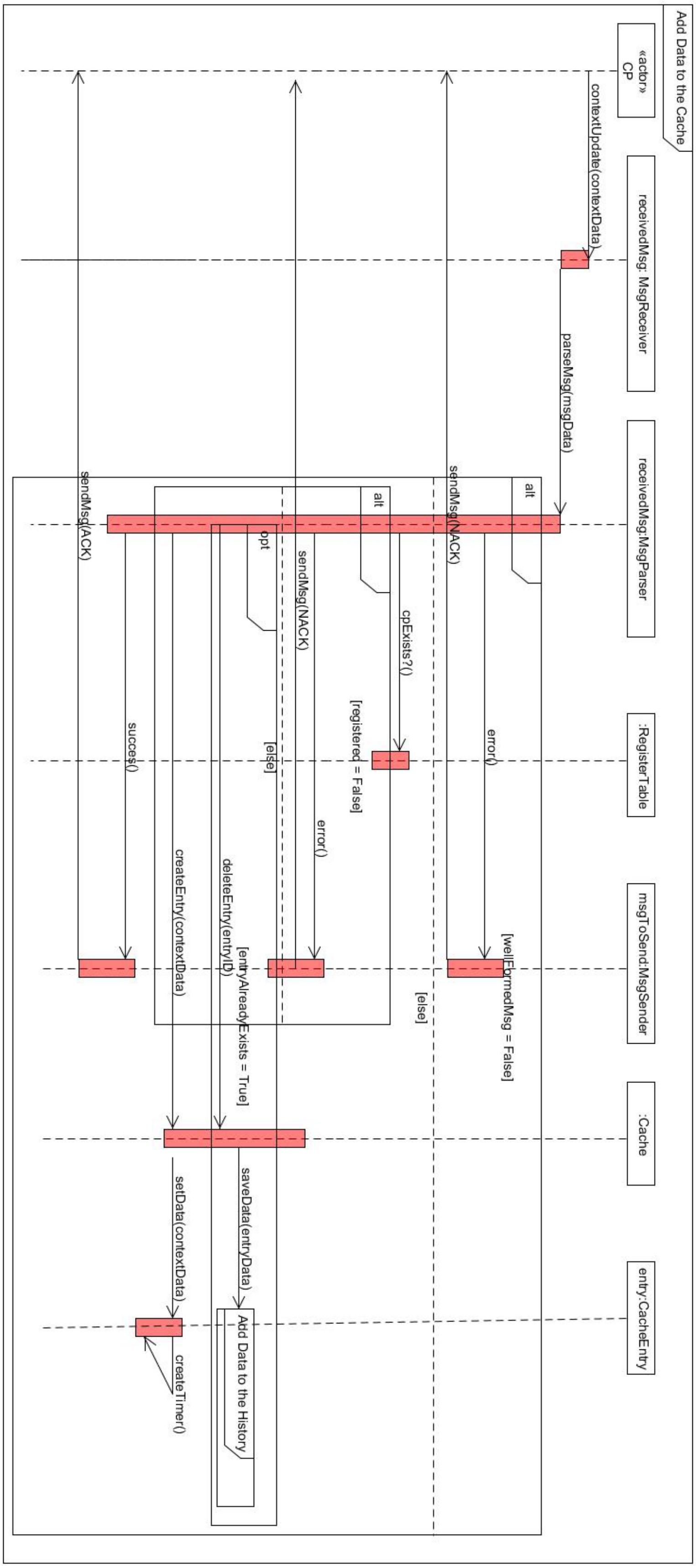


Figure 4.12: Add Data to the Cache

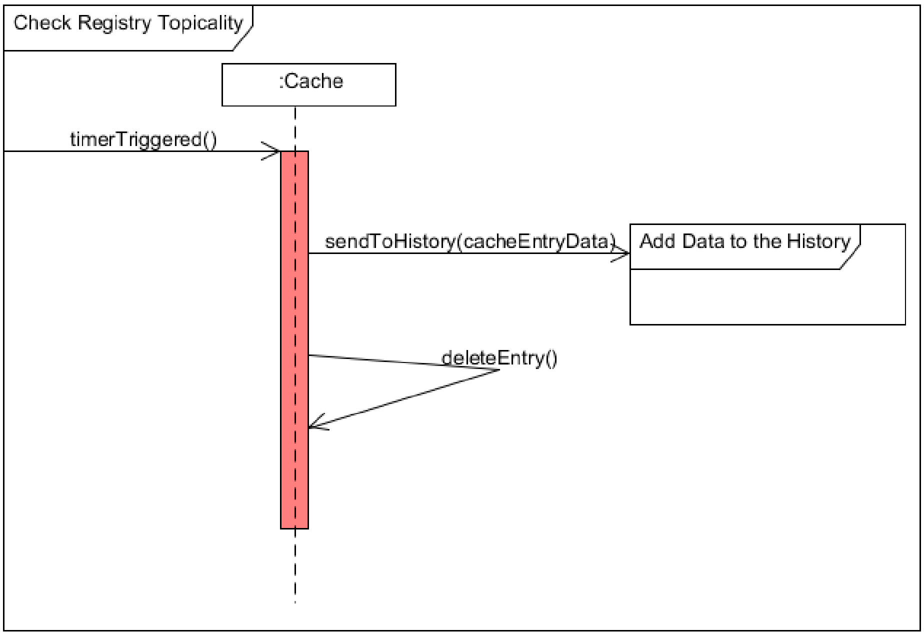


Figure 4.13: Check Validity of Context Data

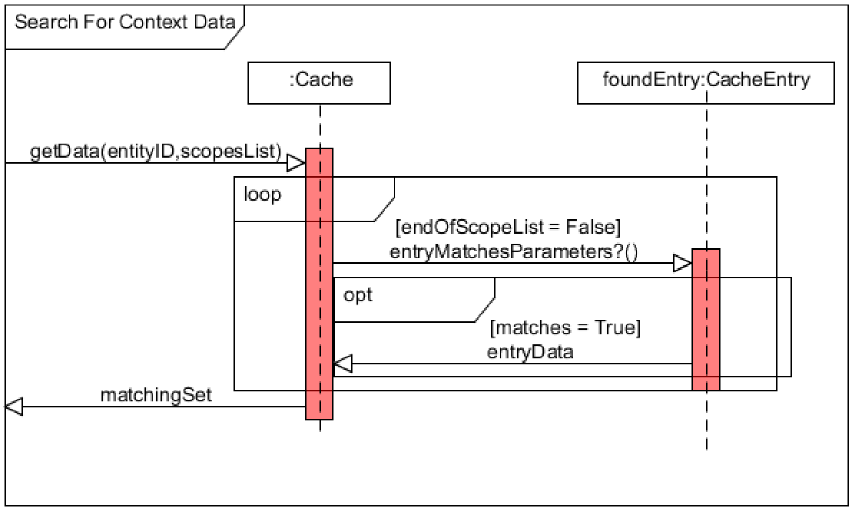


Figure 4.14: Search For Context Data

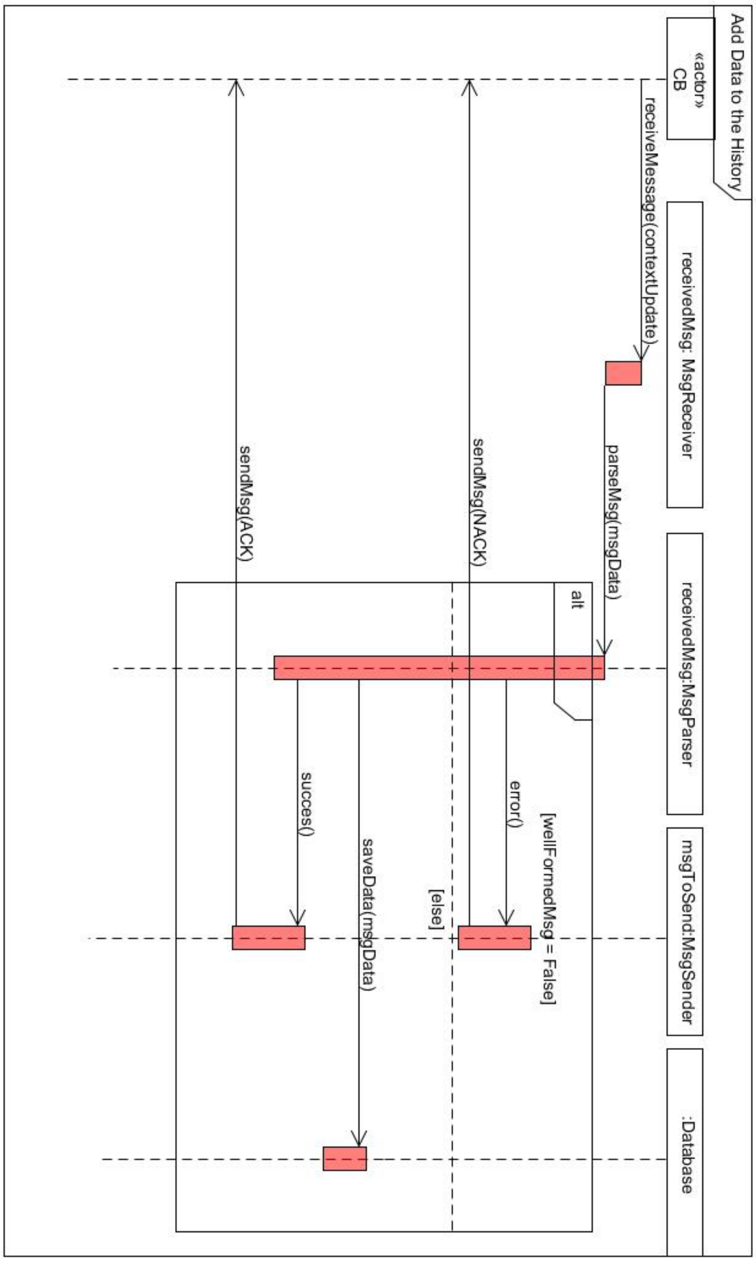


Figure 4.15: Add Data to the History

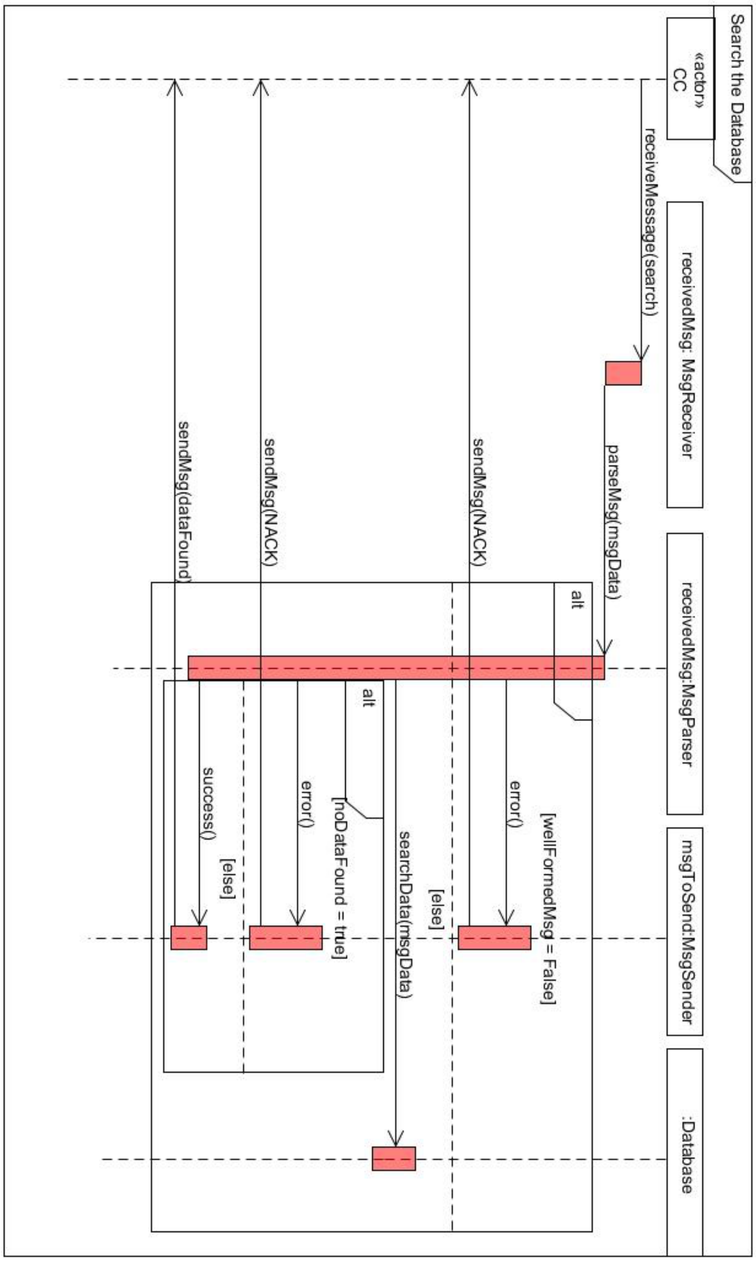


Figure 4.16: Search The Database

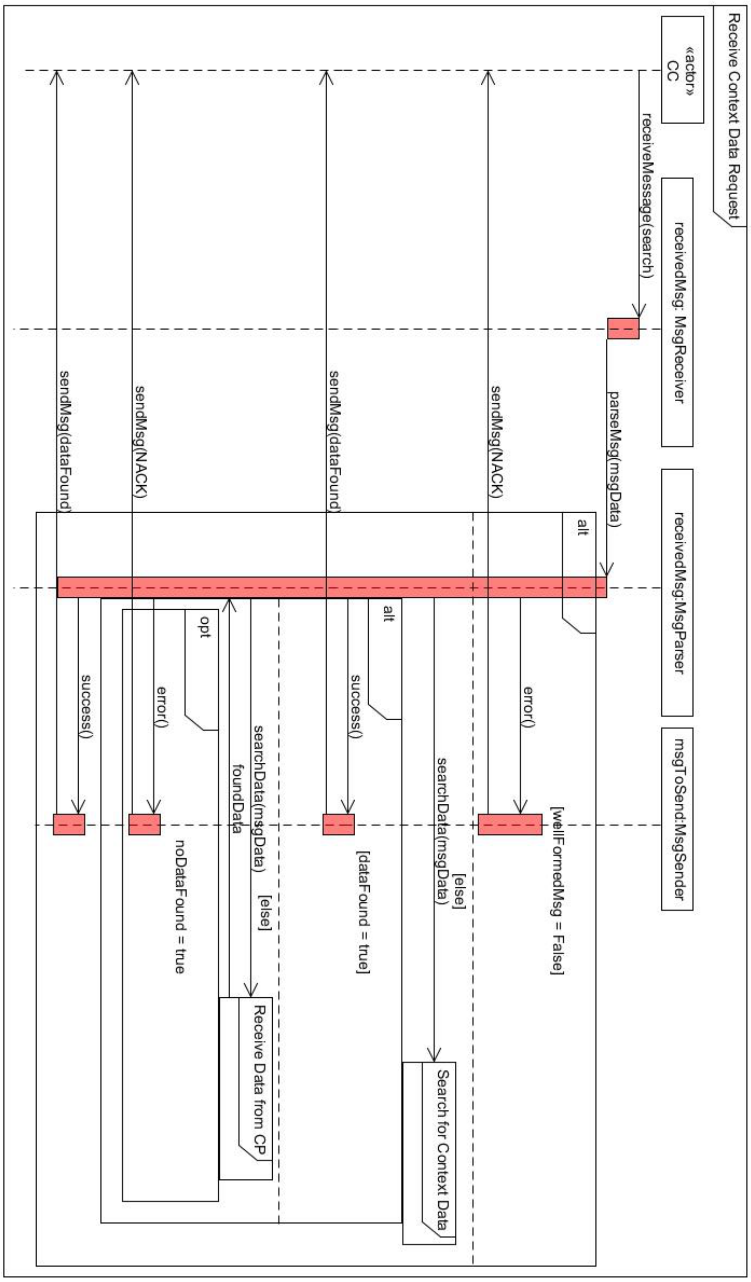


Figure 4.17: Receive Context Data Request

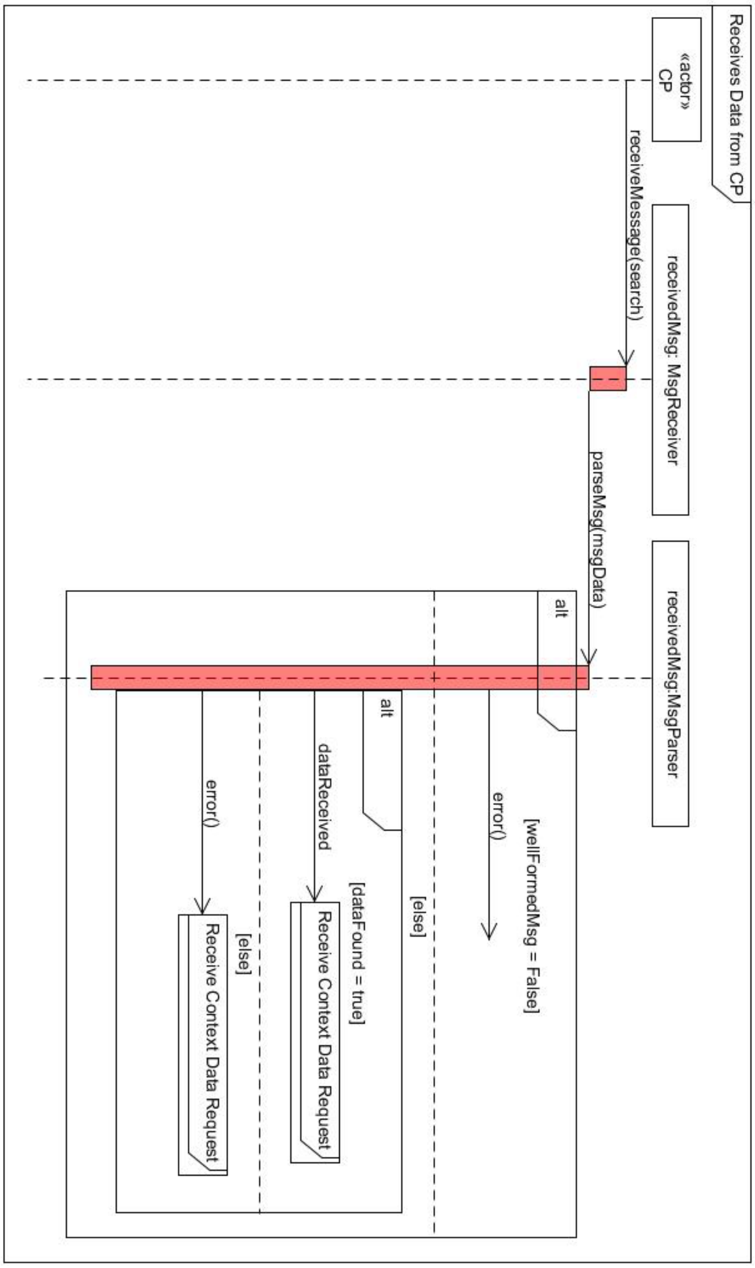


Figure 4.18: Receives Data from CP

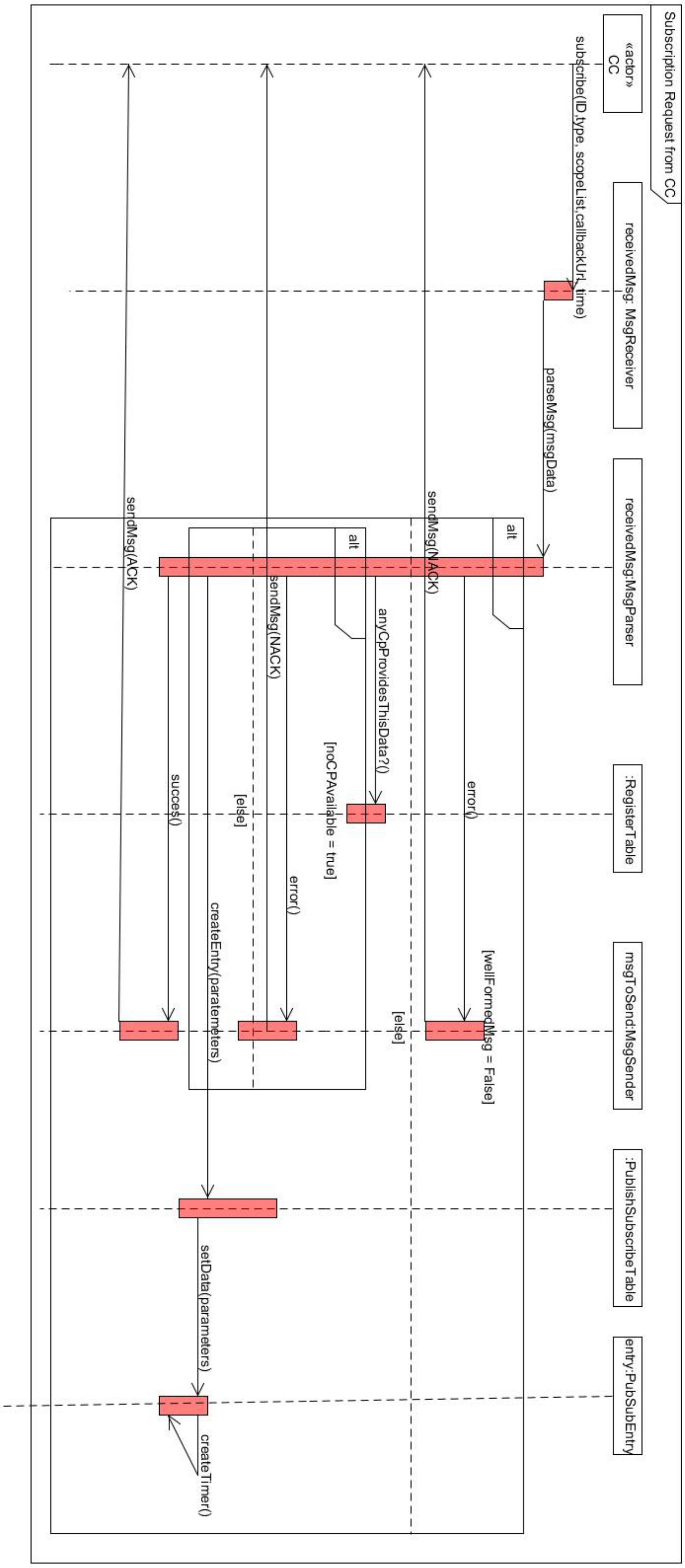


Figure 4.19: Subscription Request from CC

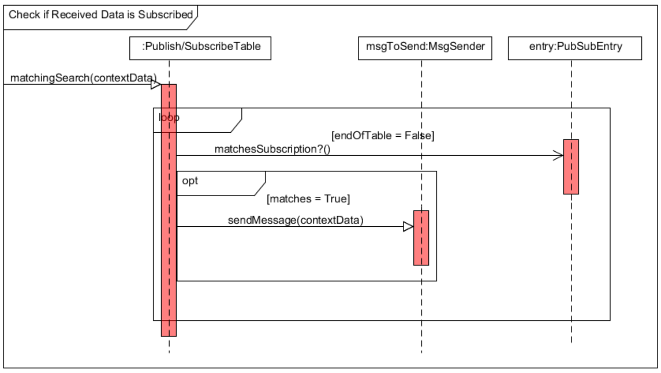


Figure 4.20: Check if Received Data is Subscribed

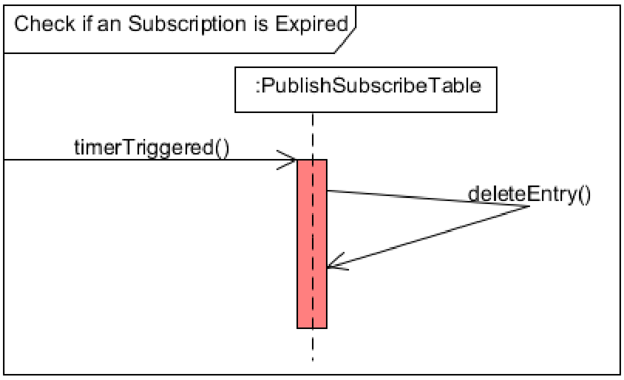


Figure 4.21: Check if an Subscription is Expired

# 5 IMPLEMENTATION AND TESTING OF THE CONTEXT BROKER

This chapter will discuss the implemented prototype of the Context Broker. The prototype follows the design presented in the previous chapter. Every module of the system has specific characteristics which will be detailed. The technical details of the development will be presented, and discussed. Finally, the test environment and results will be presented.

## 5.1 HTTP REST Interface

REST (Representational State Transfer) is a software architectural style used in distributed hypermedia systems, such as World Wide Web. The basic idea is that the client, when not interacting with the server, is at rest and does not create any processing or storage load in the server. When the client wants to transition between states, it sends requests to the server. While there are one or more requests oustanding, the client is considered to be transitioning between states. All the communication is stateless, every interaction containing all information necessary for the server to respond to the request. Clients and servers are separated by a uniform interface, and act completely independently. Servers must be addressable by URLs. More information can be found in [Fielding. 2002].

The use of HTTP comes naturally when considering a REST interface, since it already has all the mechanisms and functions to operate with stateless communication and URLs. It is a widely used protocol. Most if not all CCs and CPs already support HTTP, so extending them to communicate with the Broker should not be difficult. Every functionality that a Broker needs to provide is going to have a HTTP REST interface for easy access. If data must be sent to the Broker, a POST HTTP message should be used. If an URL must be accessed, a HTTP GET message should be used.

The CB uses HTTP Servlets to handle requests from CPs and CCs. Servlets are Java objects which handle HTTP Requests. They provide a high level abstraction, so the programmer does not need to know the low level details of HTTP to be able to respond to requests. They receive a request, do some processing, and send a response. In this project, the processing is done by Session Beans, discussed in the next section.

All message types refered here come from the ContextML, discussed in Chapter 2. The ContextML messages the Broker interacts with (be it by sending or receiving one) are validated against the ContextML XML Schema. Only well formed message are accepted. The XML data must be complete and properly formed.

Some interfaces only accept HTTP POST messages (the ones where a ContextML message is expected) and some only accept HTTP GET messages (the ones where only a command to the Broker is expected). Before any processing is done, the type of the HTTP request is checked, and if it is the wrong type for that interface, it is discarded.

All ACK and NACK messages indicated in the design (see previous chapter) are of type <ctxResp>. NACK messages contain the error code and a string explaining the cause of the error. If the processing occur without any error, an ACK message is sent (except in interfaces that return some kind of data - they have specific responses). If any error occurs during the processing, a NACK message is sent.

## 5.2 Development Environment

This project was developed using the Java programming language [JAVA. 2010]. More specifically, the platform Java EE (Java Platform, Enterprise Edition) [JAVAEE. 2010]. This platform was created for server programming in Java. It is widely used and provides a great support for the developer. It was designed for the creation of distributed, multi-tier software, running on application servers.

JBOSS [JBOSS. 2010] was used as the application server. It is cross-platform, supports Java EE and provides many functionalities. The IDE used was NetBeans [NETBEANS. 2010], and all the development was done in Suse Linux [SUSE. 2010].

Java EE has two important components used in this project. The first one is Session Beans. They represent one client session in the server. The ones used in the project were stateless, due to the stateless nature of the REST interfaces. They provide a good high-level and easy to program framework for server programming.

The second important component was the Entity Bean. They provide a way so that, using only Java, the programmer can interact with databases and persist any data it finds necessary. The Java Persistence API (JPA) manages all the necessary operations to store and change data in the DB. The database used was HSQLDB [HSQLDB. 2010], and to handle the object-relational database mapping, Hibernate [HIBERNATE. 2010] was used.

## 5.3 Modules

In this section, all the modules of the Context Broker will be detailed. The modules implemented the design of the previous chapter. They all have some form of interface, so the CCs and/or CPs can access their functions.

### 5.3.1 Cache

The Cache was implemented as a static class. A static class in Java is a class with all its methods and variables marked as static. The Cache is instantiated only once, effectively creating only one Cache table that is accessed by all threads of the CB. Since more than one thread can access the data, concurrent access control is used (in Java, through the use of the keyword synchronized in the method's signature). Only one thread can access the Cache at a given time.

When a CP sends data to the CB, it must specify the period of time in which the data is valid (using the tags <timestamp> and <expires> in the <ctxEls> message type). Every entry in the Cache has a timer associated with it, that runs up when the time indicated in <expires> is up. Outdated data is sent to the Context History.

A CP can send new data at any time. If a newly received context data updates a cache entry that is still valid, the old entry is sent to the History and deleted. The new data is then inserted.

#### *5.3.1.1 The **getContext** interface*

In order to access the data in the Cache, a Consumer must access the URL `http://<broker+ip>/ContextBroker/getContext`. Since the Broker must provide the most generic interface possible, the searching parameters are rather simple:

- **scopeList :** a list of scopes, separated by commas (for instance, "position,civilAddress,weight");
- **entity :** a certain entity ID (for instance, "Marcos");
- **type :** a certain entity type (for instance, "username").

All parameters must be given. If the consumer needs a more filtered set of results, then it must contact the CP (or CPs) that can provided the requested data directly.

First, the CB searches the Cache for entries that match the searching criteria. If it can not find any for the requested [scope,entityId,entityType], it searches the RegisterTable for CPs that can provide data for the [scope,entityType] given. If one is found, then the CB asks if it has any data related to the entity with ID [entityID].

The response given is a `<ctxEls>` with all the cache entries that match the search criteria.

The possible errors are: lack of parameters or wrong format, or no results found.

#### *5.3.1.2 The **contextUpdate** interface*

All CPs are required to regularly send updated data to the CB. This data is stored in the Cache. In order to do so, the CP must send a `<ctxEls>` message to the URL `http://<broker+ip>/ContextBroker/contextUpdate`.

After the CB receives the message, it checks if the CP that sent the data has already registered in the Register Table (that is, sent at least one provider advertisement). If it has not, the data is discarded. If the data is new, the cache entry is created and inserted in the Cache. If there is already data for that entity and scope, the entry is replace. If there are any subscribers to this data, the CB forwards it to the interested Consumers.

One important point to elucidate: there is no global clock in the system. It is very likely that the internal clock of a CP does not match the internal clock of the Broker. When the CB receives a context update where the time tags (`<timestamp>` and `<expires>`) are both in the future (in relation to the CB's clock), then they are synchronized with the CB's clock. If the data was generated less than a minute before the CB's clock, then this synchronization is also done.

The possible errors are: the time tags were set wrong (either they have the same value, or `<expires>` occurs before `<timestamp>`), or the CP was not registered in the Broker.

### 5.3.2 Register Table

The Register Table was implemented as a static class. During the running of the Broker, only one Register Table will be instanciated. Besides that, it allows for concurrent access and controls it. When a new entry in the table is created, a timer is associated with it. All CPs must regularly re-advertise their capabilities. If the timer associated with an entry runs up, the Broker tries to contact the Provider. If the Provider responds, the timer is reset. If not, the entry is marked as related to a disconnected Provider. Every time a Provider re-advertises, the timer associated with its entry is restarted.

#### *5.3.2.1 The **getContextProviders** interface*

If a Consumer wants to get a list of CPs that provide a certain context data, it must access the interface URL `http://<broker+ip>/ContextBroker/getContextProviders`. This interface's parameters are:

- **scope :** a certain scope (for instance, "position");
- **entity :** a certain entity type (for instance, "username").

At least the parameter **scope** must be provided. The response given is a `<ctxPrvEls>` message, listing all the Providers found that can provide data for that entity type and scope.

The possible errors are : no results found and scope not provided.

#### *5.3.2.2 The **providerAdvertising** interface*

One of the most important tasks of the Context Broker (and what defines it as a broker) is the ability of indicating for any Consumer all the CPs in the system. The providerAdvertising interface allows a CP to inform the CB of its existance and its capabilities. The Provider sends a `<ctxAdvs>` message to the URL `http://<broker+ip>/ContextBroker/providerAdvertising` . It is assumed that the CB's IP will be known in the system. No discovery protocol was implemented : a new CP has the Broker's IP and sends the advertisement when it is ready.

### 5.3.3 History

The Context History module was implemented as a separate program. The CH is a special kind of Context Provider. At the same time that its functionality is essential to the Broker's functioning, all others components have access to its data. The main difference from the other CPs is that there is one interface (the /add interface) that only accepts messages from the Broker. Only the CB can be sure that a certain context information is outdated, since it is the only component that needs to guarantee its data is up-to-date. Besides that, it is expected from the CB to be a powerful machine, while the CH is just another CP. So instead of possibly overloading the CH, only the CB handles adding data to the History. The CB is already expected to handle great quantities of data.

Using JPA, an entity called History was created. The attributes of that entity class are the same as the entries in the cache (basically, the data present in any `<ctxEl>`). Inside the program, all the data is handled in the format of Java objects. The JPA and Hibernation manage the translation to the database format and interact with the database.

#### *5.3.3.1 The **add** interface*

When the Broker wants to add outdated data to the Context History, it sends a `<ctxEls>` message with only one `<ctxEl>` (the outdated cache entry data) to the URL `http://<broker+ip>/History-war/add`. The CH parses the message and creates the entry in the database. Only messages from the Context Broker are allowed.

#### *5.3.3.2 The **search** interface*

Accessing the URL `http://<history+ip>/History-war/search`, any Consumer or Provider can search the database for relevant data. The message return is of the type `<ctxEls>`. The only difference from a common `<ctxEls>` is that, for each

`<ctxEl>`, in its `<dataPart>`, there is a parameter `<par n=\"history\">true</par>` . The addition of this `<par>` guarantees one can determine that the origin of the ctxResp is the Context History.

The Context History does not accept a search with no parameters, at least one has to be given. Any combination of the parameters is allowed.

The possible parameters are:

- **cpID** : the ID of a Context Provider;
- **entityId** : a certain entity ID;
- **entityType** : a certain entity type;
- **scope** : a certain scope;
- **begin** : a time parameter for the initial time;
- **end** : a time parameter for the end time.

The time parameters (begin and end) require some further explaining.

First, this is an example of a valid input time :

- 2009-11-06T15:05:00%2B01:00

The %2B is the encoding for the plus sign. The plus sign can not be written in a URL, because it is transformed into a space. Initially, the day is inserted, in the order YEAR-MONTH-DAY. Then the time is inserted Thour:minutes:seconds. The information after the plus sign is the time zone of the time provided, in relation to Greenwich Mean Time (GMT).

There are three possible uses:

- **Only begin is provided:** return only data valid before the given time;
- **Only valid is provided:** return only data valid after the given time;
- **Both are provided:** return data valid in the period of time specified.

Some examples of uses:

- `/History-war/search?scope=poaCapabilities&begin=2009-11-06T15:05:00%2B01:00`
- `/History-war/search?cpID=CP&entityID=me&entityType=wnn&scope=poaCapabilities`
- `/History-war/search?scope=poaCapabilities`
- `/History-war/search?cpID=CP&scope=poaCapabilities`

There are four possible errors in an attempted search: no parameters where provided, the wrong parameters where provided, a database error or no results where found.

### 5.3.4 Publish/Subcribe System

The Publish/Subscribe system allows for asynchronous delivering of context data to subscribers. A Consumer needs to access the URL `http://<broker+ip>/ContextBroker/subscribe` in order to subscribe to a particular data. When any data matching the subscription criteria arrives at the Broker, the CB forwards it to all subscribers interested in that information. The parameters are:

- **entity** : a certain entity ID;
- **type** : a certain entity type;
- **scopeList** : a list of scopes, separated by ',' (for instance, "position,civilAddress,weight");
- **callbackUrl** : the URL to where the data must be sent when something matching the criteria arrives;
- **time** : the amount of minutes the subscription is valid. It must be an integer number bigger than zero.

The time parameter must be bigger than zero, and integer. The reason why is that it represents how many minutes the subscription is valid. There is no possiblity of subscribing for a context indefinitely (although, the Consumer could subscribe for a long period of time).

There is no obstacle in the architecture for the implementation of a subscription using a numerical threshold (for instance, "scope X > 30.99" ). The time parameter was chosen so because then the interface would be as generic as possible. The Consumer can filter any data sent to it by the subscription system, and may or may not apply a numerical threshold.

It is the Consumer's responsibility to renew a subscription that became invalid. There is a timer associated to each subscription: after that timer runs up, the subscription becomes invalid and the Broker does not try to contact the consumer through the callbackURL.

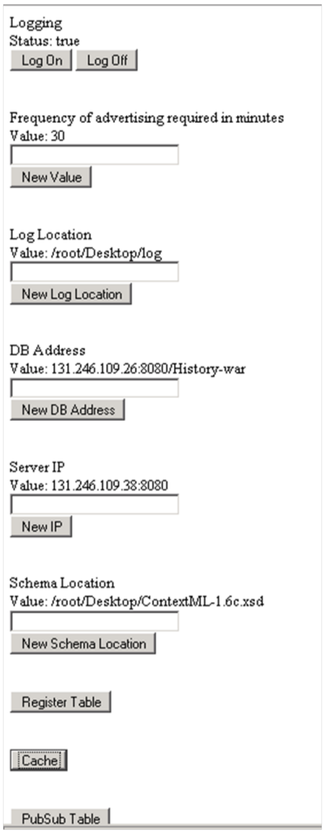
Just like the Cache and the Register Table, the Publish/Subscribe table was implemented as a static class with concurrent access control.

The possible errors are: no CPs found that can provide that data, lack of parameters or wrong format for the time parameter.

### 5.3.5 Configuration Page and getName interface

The CB has an internal configurations file internally, where information necessary for it to run is stored (for instance, the location of the XML Schema file for the ContextML). A simple web page was created to allow changes in the configuration settings during the execution of the Broker. It can be accessed in the URL `http://<broker+ip>/ContextBroker`. It is important to emphasize that the config page can only be accessed through either the machine the CB is running on, or through a machine with a valid IP (currently, IPs belonging to the Wireless Communications and Navigation group in the TUKL).

The config page can also display the content of all the tables (Register Table, Cache and Publish/Subscribe Table). Figure 5.1 shows the configuration page. Figure 5.2 shows the Register Table. Figure 5.3 shows the Cache.



Logging
Status: true

Frequency of advertising required in minutes
Value: 30

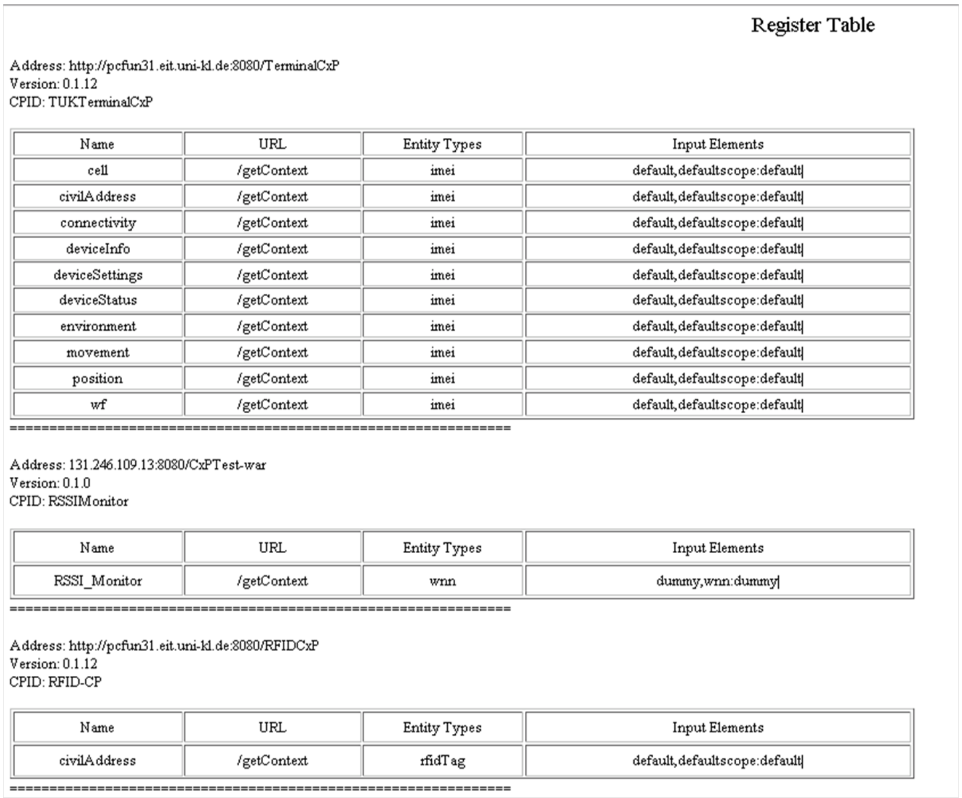
Log Location
Value: /root/Desktop/log

DB Address
Value: 131.246.109.26:8080/History-war

Server IP
Value: 131.246.109.38:8080

Schema Location
Value: /root/Desktop/ContextML-1.6c.xsd

Figure 5.1: Configuration Page



Register Table

Address: http://pcfun31.eit.uni-kl.de:8080/TerminalCxP
Version: 0.1.12
CPID: TUKTerminalCxP

| Name | URL | Entity Types | Input Elements |
|---|---|---|---|
| cell | /getContext | imei | default,defaultscope:default\| |
| civilAddress | /getContext | imei | default,defaultscope:default\| |
| connectivity | /getContext | imei | default,defaultscope:default\| |
| deviceInfo | /getContext | imei | default,defaultscope:default\| |
| deviceSettings | /getContext | imei | default,defaultscope:default\| |
| deviceStatus | /getContext | imei | default,defaultscope:default\| |
| environment | /getContext | imei | default,defaultscope:default\| |
| movement | /getContext | imei | default,defaultscope:default\| |
| position | /getContext | imei | default,defaultscope:default\| |
| wf | /getContext | imei | default,defaultscope:default\| |

==============================================================

Address: 131.246.109.13:8080/CxPTest-war
Version: 0.1.0
CPID: RSSIMonitor

| Name | URL | Entity Types | Input Elements |
|---|---|---|---|
| RSSI_Monitor | /getContext | wnn | dummy,wnn:dummy\| |

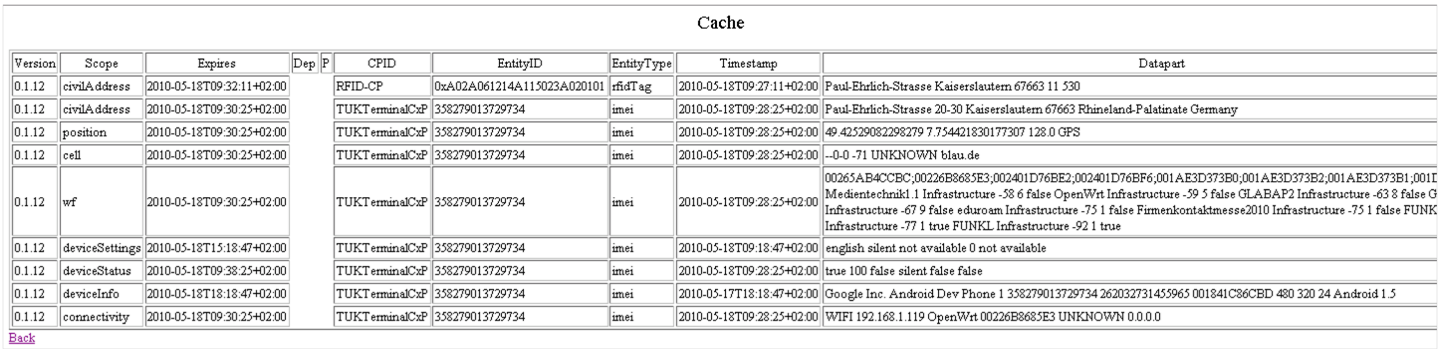
==============================================================

Address: http://pcfun31.eit.uni-kl.de:8080/RFIDCxP
Version: 0.1.12
CPID: RFID-CP

| Name | URL | Entity Types | Input Elements |
|---|---|---|---|
| civilAddress | /getContext | rfidTag | default,defaultscope:default\| |

==============================================================

Figure 5.2: Register Table



Cache

| Version | Scope | Expires | Dep | P | CPID | EntityID | EntityType | Timestamp | Datapart |
|---|---|---|---|---|---|---|---|---|---|
| 0.1.12 | civilAddress | 2010-05-18T09:32:11+02:00 | | | RFID-CP | 0xA02A061214A115023A020101 | rfidTag | 2010-05-18T09:27:11+02:00 | Paul-Ehrlich-Strasse Kaiserslautern 67663 11 530 |
| 0.1.12 | civilAddress | 2010-05-18T09:30:25+02:00 | | | TUKTerminalCxP | 358279013729734 | imei | 2010-05-18T09:28:25+02:00 | Paul-Ehrlich-Strasse 20-30 Kaiserslautern 67663 Rhineland-Palatinate Germany |
| 0.1.12 | position | 2010-05-18T09:30:25+02:00 | | | TUKTerminalCxP | 358279013729734 | imei | 2010-05-18T09:28:25+02:00 | 49.42529082298279 7.754421830177307 128.0 GPS |
| 0.1.12 | cell | 2010-05-18T09:30:25+02:00 | | | TUKTerminalCxP | 358279013729734 | imei | 2010-05-18T09:28:25+02:00 | --0-0 -71 UNKNOWN blau.de |
| 0.1.12 | wf | 2010-05-18T09:30:25+02:00 | | | TUKTerminalCxP | 358279013729734 | imei | 2010-05-18T09:28:25+02:00 | 00265AB4CCBC;00226B8685E3;002401D76BE2;002401D76BF6;001AE3D373B0;001AE3D373B2;001AE3D373B1;001D<br>Medientechnik1.1 Infrastructure -58 6 false OpenWrt Infrastructure -59 5 false GLABAP2 Infrastructure -63 8 false G<br>Infrastructure -67 9 false eduroam Infrastructure -75 1 false Firmenkontaktmesse2010 Infrastructure -75 1 false FUNK<br>Infrastructure -77 1 true FUNKL Infrastructure -92 1 true |
| 0.1.12 | deviceSettings | 2010-05-18T15:18:47+02:00 | | | TUKTerminalCxP | 358279013729734 | imei | 2010-05-18T09:18:47+02:00 | english silent not available 0 not available |
| 0.1.12 | deviceStatus | 2010-05-18T09:38:25+02:00 | | | TUKTerminalCxP | 358279013729734 | imei | 2010-05-18T09:28:25+02:00 | true 100 false silent false false |
| 0.1.12 | deviceInfo | 2010-05-18T18:18:47+02:00 | | | TUKTerminalCxP | 358279013729734 | imei | 2010-05-17T18:18:47+02:00 | Google Inc. Android Dev Phone 1 358279013729734 262032731455965 001841C86CBD 480 320 24 Android 1.5 |
| 0.1.12 | connectivity | 2010-05-18T09:30:25+02:00 | | | TUKTerminalCxP | 358279013729734 | imei | 2010-05-18T09:28:25+02:00 | WIFI 192.168.1.119 OpenWrt 00226B8685E3 UNKNOWN 0.0.0.0 |

Figure 5.3: Cache

The getName interface (URL `http://<broker+ip>/ContextBroker/getName` only returns a simple `<ctxResp>` with basic information about the Broker. It should be used by CPs and CCs to check if the Broker is online and running correctly. Accessing this interface guarantees that the CC or Cp will not send or alter any data in the CB.

## 5.4 Testing

This project main goal was to design and implement the Context Broker. So all the testing done was focused in ensuring that the Broker operated correctly. The ambient in which the tests occurred were ideal. All the machines involved were connected in a local network.

The tests objectives were:

- Ensure that the Broker was operating correctly;
- Ensure stability (operating without any errors for long periods of time);
- Ensure no big delays in processing requests;
- Ensure that the data in the Cache was always the most up-to-date;
- Ensure that the CPs marked as offline were really offline;
- Ensure that all functional requirement were properly fulfilled.

The test setting was as follows:

- Two CPs were used. They were the CPs being developed by the other members of the research group.
  - **Wireless Access CxP :** this CP provides data related to the capabilities of a wireless point of access.
  - **Terminal Sensors CxP :** provides sensor data from a terminal (in this case a cell phone operating with Android).
- The Broker operated in one machine.
  - **Machine specification :** Intel Core 2 Duo CPU 2.4GHz, 2.0 GB RAM
  - **Operating System :** Suse Linux 10.3

  - **Application Server :** JBOSS Application Server release 5.1.0.GA

- The Context History operated in a dedicated machine, separated from the Broker.

The tests were sucessful, meeting all goals. The Broker was able to operate two weeks without any errors or major delays. The average response time was less than 100ms (in ideal conditions, of course). Both the CPs used had fairly regular context updates (one every minute), so the amount of data generated with time was considerable, but the Broker did not slow down or crashed.

During the running of the Broker, simple programs were created to emulate Consumers making punctual requests (subscribing, requesting data, etc..). All those functions operated correctly and did not crash the system, in the tests made. Also, despite all the processing that occurs concurrently with the Consumer's requests (contexts update, sending data to the History, provider advertisements, timers running up), there was no noticeable delay between all the requests. This is important because the Consumers should not wait too long for content.

# 6 CONCLUSION

This thesis presented the design and implementation of a Broker for a Service-Oriented Context Management and Distribution Architecture. All the relevant concepts were presented, the architecture was described, and the design, implementation and testing processes were shown.

Initially, the theoretical foundations for this work were presented. Context and context awareness are areas that have been researched for a long time, providing a lot of material to work with. In order to properly situate this project, a brief history of the research was presented. all the necessary definitions were presented, and also common solutions and approaches used in context modelling and context management.

The definition of context and context awareness used is the one proposed by [Dey. 2000], which is the most general and complete one found in the literature. The architecture presented in this work uses a context server, resides in a networked system and has a markup language called ContextML as a method for context modelling.

The architecture this Broker operates in determines which features it should have. First, the basic definitions such as scope, entity and their relationship were presented. They are simple and useful definitions used by all components. The main elements of the markup language used, ContextML, were shown. It is a complete language in the sense that not only does it describe context information, but it also helps in the functioning of the system, such as Context Provider Advertisement. Finally, each component and its respective responsibilities in the overall system was described, and the manner of communication between components was presented.

With the architecture properly defined, getting all the requirements and designing the Broker was a natural next step. The design provides a general and complete view of all the necessary modules and their behavior. UML was used to design this project, and the diagrams and information presented followed its specification.

The implementation process was described in the next chapter. All the system uses HTTP interfaces and messages. All the consumers and providers have web interfaces, and HTTP provides an easy way for them to comunicate with the Broker. There is no need for them to install new modules or work with specific protocols. The technology used to implement was J2EE and Servlets. Servlets were used because they give a good high level way of implementing web services. J2EE was used because it is a solid and complete platform for server programming. Both have good implementations, plus good applications servers such JBOSS (the one used in this work) are available.

The final objective of this work was met with the design and implementation of a stable Broker. The tests done showed that it provides all the features presented and it can operate for long periods of time without crashing. The testing conditions were ideal, but that was in order to validated the functioning of the Broker. Exposing the Broker to more chaotic and unreliable environments is a logical next step.

## 6.1 Future Work

This work lays the ground for the basic behavior for the Context Broker. The features presented here define what a broker should be and do. When considering where to go from here, two aspects stand out : scalability and context discovery.

### 6.1.1 Context Broker Federation

The Context Broker is the central point of the whole architecture. It is crucial that it remains functional and able to handle great amounts of requests. The first and simplest solution for that problem is to have a machine with a lot of processing power and storage capability to operate as the Broker. However, as more consumers and providers enter the system, that machine will be enough. The next logical step is a server farm, a cluster that can be expanded as needed (or as the costs allow it).

Even a giant cluster will not be able to properly and efficiently handle multiples requests from around the world. The best solution would be having multiple Brokers spread around the world, each one dealing with requests from its own area. The consumers and providers perceive the Broker as one interface. They are not required to keep lists of Brokers and their locations, so a consumer could ask in Japan for data that the Broker in the US has in its Cache.

Therefore, a big and challenging next step to the work proposed here would be to design a federation of Context Brokers, geographically distributed, that operate as one interface, seemlessly to all the other components of the architecture. Protocols for searching data in this network, how to maintain the network operating (with no Broker unreachable), how to deal with a new Broker, how the Brokers will find each other, among many others are questions that this possible future work will have to answer.

### 6.1.2 Entity Resolution

The system, as it is presented now, has a very simplistic way to deal with entity/scope information. Every Context Provider directly informs which entities and scopes it deals with.

A significant improvement for that solution is to give the Broker the ability to detect relationship between entities and scopes. The Broker would not only detect when entities are connected, but to which degree that conection exist.

The best example is entity mobile and entity username. Both have the scope location, which represent the geographic coordinates for the mobile device and for the user. It is uncommon for a person to have something that pin points his/her location. However, it is very common for someone to have a cell phone. So the location of the cell phone can, to a certain degree, provide a good estimative for the location of the user.

Another example would be the civil address for a person. If the address of someone who is married is unknown, the address from his/her partner could be used with a certain degree of trust.

Both discovering the relationships and determining the degree of trust are very abstract and theoretical problems. The idea is to give the Broker some intelligence, so the information a consumer can receive is more complete and richer in details.

# REFERENCES

[Dey. 2000] Dey A.K., Abowd G.D., **Towards a better understanding of context and context-awareness**. Proceedings of the Workshop on the What, Who, Where, When and How of Context-Awareness. ACM Press, New York, 2000.

[Schilit. 1994] Schilit B., Theimer M., **Disseminating Active Map Information to Mobile Hosts**. IEEE Network, 8(5) (1994) 22-32.

[Brown. 1997] Brown P.J., Bovey J.D.,Chen X., **Context-Aware Applications: From the Laboratory to the Marketplace**. IEEE Personal Communications, 4(5) (1997) 58-64.

[Ryan. 1997] Ryan N., Pascoe J., Morse D., **Enhanced Reality Fieldwork: the Context-Aware Archaeological Assistant**. Computer Applications in Archaeology (1997)

[Pascoe. 1998] Pascoe J., **Adding Generic Contextual Capabilities to Wearable Computers**. 2nd International Symposium on Wearable Computers (1998) 92-99.

[Dey. 1999] Dey A.K., Abowd G.D., Wood A., **CyberDesk: A Framework for Providing Self-Integrating Context-Aware Services**. Knowledge-Based Systems, 11 (1999) 3-13.

[Hull. 1997] Hull R., Neaves P., Bedford-Roberts J., **Towards Situated Computing**.

1st International Symposium on Wearable Computers (1997) 146-153.

[Franklin. 1998] Franklin D., Flaschbart J., **All Gadget and No Representation Makes Jack a Dull Environment**. AAAI 1998 Spring Symposium on Intelligent Environments, Technical Report SS-98-02 (1998) 155-160.

[Ward. 1997] Ward A., Jones A., Hopper A., **A New Location Technique for the Active Office**. IEEE Personal Communications 4(5) (1997) 42-47.

[Rodden. 1998] Rodden T., Cheverst K., Davies K., Dix A., **Exploiting Context in HCI Design for Mobile Systems**. Workshop on Human Computer Interaction with Mobile Devices (1998).

[Chen. 2004] Chen H., (2004) **An Intelligent Broker Architecture for Pervasive Context-Aware Systems**, PhD Thesis, University of Maryland, Baltimore County.

[Winograd. 2001] Winograd T., 2001, **Architectures for Context**. Human-Computer Interaction, 16(2), 401

[Dey. 2001] Dey A., Salber D., Abowd G., 2001. **A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications**.

[Chen. 2003] Chen H., Finin, T., Joshi A., 2003. **An Intelligent Broker for Context-Aware Systems**, Adjunct Proceedings of Ubicomp 2003

[Strang. 2004] Strang T., Linnhoff-Popien C., 2004. **A Context Modeling Survey**. In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England.

[CCPP. 2010] **Composite Capabilities/Preferences Profile Home Page**, http://www.w3.org/Mobile/CCPP. Acessed in January 21, 2010.

[Hofer et al. 2003] Hofer T. et al., 2003. **Context-awareness on mobile devices - the hydrogen approach**. In System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on.

[Sheng. 2005] Sheng Q., Benatallah B., 2005. **ContextUML: a UML-based modeling language for model-driven development of context-aware Web services**. In Mobile Business, 2005. ICMB 2005. International Conference on. pp. 206-212.

[Henricksen et al. 2005] Henricksen K. et al., 2005. **Middleware for Distributed Context-Aware Systems**. In On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE. pp. 846-863.

[Baldauf. 2007] Baldauf M., Dustdar S., Rosenberg F., 2007. **A survey on context-aware systems**. International Journal of Ad Hoc and Ubiquitous Computing, 2(4), 263-277.

[Wang et al. 2004] Wang, X. et al., 2004. **Ontology based context modeling and reasoning using OWL**. In Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on. pp. 18-22.

[UML. 2010] Object Management Group. **Unified Modelling Language**. Available at: <http://www.uml.org/>. Visited on: July 2010.

[C-CAST. 2010] ICT group. **Project Context Casting**. Available at: <http://www.ict-ccast.eu/>. Visited on: July 2010.

[Fielding. 2002] Fielding Roy T., Taylor, Richard N., 2002, **Principled Design of the Modern Web Architecture**. ACM Transactions on Internet Technology (TOIT) Association for Computing Machinery 115-150.

[JAVA. 2010] Sun Microsytems. **Java Programming Languag**e. Available at: <http://java.sun.com/>. Visited on: July 2010.

[JAVAEE. 2010] Sun Microsytems. **Java Enterprise Edition**. Available at: <http://java.sun.com/javaee>. Visited on: July 2010.

[JBOSS. 2010] Red Hat. **JBoss Application Server**. Available at: <http://www.jboss.org>. Visited on: July 2010.

[NETBEANS. 2010] Oracle Corporation. **NetBeans IDE**. Available at: <http://netbeans.org>. Visited on: July 2010.

[SUSE. 2010] Novell. **openSUSE**. Available at: <http://www.novell.com/linux/. Visited on: July 2010.

[HIBERNATE. 2010] Red Hat. **Hibernate**. Available at: <http://www.hibernate.org>. Visited on: July 2010.

[HSQLDB. 2010] The hsql Development Group. **HyperSQL DB**. Available at: <http://hsqldb.org>. Visited on: July 2010.

# 7 APPENDIX

## 7.1 ContextML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!– edited with XMLSpy v2008 rel. 2 sp2 (http://www.altova.com) by FERNANDO GENOVA (TELECOM ITALIA SPA) –>
<xs:schema xmlns="http://ContextML/1.6c" xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://ContextML/1.6c" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:simpleType name="ResponseType">
        <xs:annotation>
            <xs:documentation>Value available for response status</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="OK"/>
            <xs:enumeration value="ERROR"/>
            <xs:enumeration value="WARN"/>
            <xs:enumeration value="TRUE"/>
            <xs:enumeration value="FALSE"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="ProviderProperty">
        <xs:annotation>
            <xs:documentation>Value available for a provider property</xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:enumeration value="id"/>
            <xs:enumeration value="url"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="par">
        <xs:annotation>
            <xs:documentation>Parameter (with name)</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:string">
                    <xs:attribute name="n" type="xs:string" use="required"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="parS">
        <xs:annotation>
            <xs:documentation>ParameterStruct contains parameters (with name)</xs:documentatio
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="par" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="n" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="parA">
        <xs:annotation>
            <xs:documentation>An array of homogenous elements: Parameters or
ParameterStructs all with the same name. The name of the array should be the plural of
the name of its childs</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:choice>
                <xs:element ref="par" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="parS" minOccurs="0" maxOccurs="unbounded"/>
            </xs:choice>
            <xs:attribute name="n" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="contextProvider">
        <xs:annotation>
            <xs:documentation>Context Provider with version and optional ID</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required"/>
            <xs:attribute name="v" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="scopeDef">
        <xs:annotation>
            <xs:documentation>Definition of the context scope</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="urlPath" type="xs:string"/>
                <xs:element name="entityTypes" type="xs:string"/>
                <xs:element ref="inputDef"/>
                <xs:element name="depUrl" minOccurs="0"/>
            </xs:sequence>
            <xs:attribute name="n" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="inputDef">
        <xs:annotation>
            <xs:documentation>Input parameters for a Context provider </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="inputEl" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:attribute name="name" type="xs:string" use="required"/>
                        <xs:attribute name="type" type="xs:string" use="required"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="dataPart">
        <xs:annotation>
            <xs:documentation>DataPart of a context element or a context response</xs:documentati
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="par" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="parS" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="parA" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="scopes">
        <xs:annotation>
            <xs:documentation>Context scopes returned by the Context Provider</xs:documentation
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="scopeDef" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="ctxResp">
        <xs:annotation>
            <xs:documentation>Context Response</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="contextProvider">
```

```
                        <xs:annotation>
                            <xs:documentation>The Context Provider that has generated the information (with ID and version)</xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="timestamp" type="xs:dateTime" minOccurs="0"/>
                    <xs:element name="entity" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>The entity this information belongs to (with ID and type)</xs:documentation>
                        </xs:annotation>
                        <xs:complexType>
                            <xs:attribute name="id" type="xs:string" use="required"/>
                            <xs:attribute name="type" type="xs:string" use="required"/>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="scope" type="xs:string" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>The scope indicates what the context information is about</xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="method" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>The requested application method </xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element name="resp">
                        <xs:complexType>
                            <xs:attribute name="status" type="ResponseType" use="required"/>
                            <xs:attribute name="code" type="xs:int" use="optional"/>
                            <xs:attribute name="msg" type="xs:string" use="optional"/>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="validity" type="xs:long" minOccurs="0"/>
                    <xs:element name="subId" type="xs:long" minOccurs="0"/>
                    <xs:element ref="dataPart" minOccurs="0"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="ctxEl">
            <xs:annotation>
                <xs:documentation>A Context Element represents context information about an entity</xs:documentation>
            </xs:annotation>
            <xs:complexType>
                <xs:sequence>
```

```
          <xs:element ref="contextProvider">
            <xs:annotation>
              <xs:documentation>The Context Provider that has generated the information (with ID and version)</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="entity">
            <xs:annotation>
              <xs:documentation>The entity this information belongs to (with ID and type)</xs:documentation>
            </xs:annotation>
            <xs:complexType>
              <xs:attribute name="id" type="xs:string" use="required"/>
              <xs:attribute name="type" type="xs:string" use="required"/>
            </xs:complexType>
          </xs:element>
          <xs:element name="scope" type="xs:string">
            <xs:annotation>
              <xs:documentation>The scope indicates what the context information is about</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="timestamp" type="xs:dateTime">
            <xs:annotation>
              <xs:documentation>Instant in time when the information was taken</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element name="expires" type="xs:dateTime">
            <xs:annotation>
              <xs:documentation>Instant in time when the information is no longer valid</xs:documentation>
            </xs:annotation>
          </xs:element>
          <xs:element ref="dataPart"/>
        </xs:sequence>
        <xs:attribute name="p" type="xs:float" use="optional"/>
        <xs:attribute name="dep" type="xs:string" use="optional"/>
        <xs:attribute name="subId" type="xs:int" use="optional"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="ctxAdv">
      <xs:annotation>
        <xs:documentation>Context Provider Advertisement describes the type of context information the provider returns </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
```

```
                    <xs:element ref="contextProvider"/>
                    <xs:element name="urlRoot" type="xs:string" minOccurs="0">
                        <xs:annotation>
                            <xs:documentation>URL root of the Context Provider
interface</xs:documentation>
                        </xs:annotation>
                    </xs:element>
                    <xs:element ref="scopes" minOccurs="0"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="ctxPrvEl">
            <xs:annotation>
                <xs:documentation>Context Provider Element describes the Context
Providers that support a scope</xs:documentation>
            </xs:annotation>
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="par">
                        <xs:complexType>
                            <xs:simpleContent>
                                <xs:extension base="xs:string">
                                    <xs:attribute name="n" type="xs:string" use="required"
fixed="scope"/>
                                </xs:extension>
                            </xs:simpleContent>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="parA" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="parS" minOccurs="0">
                                    <xs:complexType>
                                        <xs:sequence>
                                            <xs:element name="par" minOccurs="2"
maxOccurs="2">
                                                <xs:complexType>

        <xs:simpleContent>

            <xs:extension base="xs:string">

                <xs:attribute name="n" type="ProviderProperty" use="required"/>

            </xs:extension>

        </xs:simpleContent>
                                                </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="n" type="xs:string" use="required" fixed="contextProvider"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="n" type="xs:string" use="required" fixed="contextProvider"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="scopeEl">
    <xs:annotation>
      <xs:documentation>Scope element describes the scopes available on Context Broker</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="parA">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="par" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
                      <xs:attribute name="n" type="xs:string" use="required" fixed="scope"/>
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="n" type="xs:string" use="required" fixed="scope"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="contextML">
    <xs:annotation>
      <xs:documentation>ContextML document root element</xs:documentation>
    </xs:annotation>
    <xs:complexType>
```

```
<xs:choice>
    <xs:element name="sentAt" type="xs:dateTime" minOccurs="0"/>
    <xs:element name="ctxEls">
        <xs:annotation>
            <xs:documentation>A ContextML document contains one or more Context Element from different providers</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="ctxEl" minOccurs="0" maxOccurs="unbounded">
                    <xs:annotation>
                        <xs:documentation>A Context Element represents context information about an entity</xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="ctxAdvs">
        <xs:annotation>
            <xs:documentation>Contains the advertisement of Conetxt Provider features</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="ctxAdv" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element ref="ctxResp"/>
    <xs:element name="scopeEls">
        <xs:annotation>
            <xs:documentation>Contains response from the Context Broker to the getAvailableScopes method </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="scopeEl" maxOccurs="unbounded">
                    <xs:annotation>
                        <xs:documentation>Scope element describes a scope available on Context Broker</xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="ctxPrvEls">
```

```
                    <xs:annotation>
                        <xs:documentation>Contains response from the Context Broker to the getContextProviders method</xs:documentation>
                    </xs:annotation>
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element ref="ctxPrvEl" maxOccurs="unbounded"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:choice>
        </xs:complexType>
    </xs:element>
</xs:schema>
```