Design of CASP: an Open Enabling Platform for Context Aware Office and City Services

Matthias Strobbe[1], Jan Hollez[1], Gregory De Jans[1], Olivier Van Laere[1], Jelle Nelis[1], Filip De Turck[1], Bart Dhoedt[1], Piet Demeester[1], Nico Janssens[2], and Thierry Poller[2]

[1] Ghent University - IBBT - IMEC, Department of Information Technology, Gaston Crommenlaan 8 bus 201, 9050 Gent, Belgium.
[2] Alcatel-Lucent, Research & Innovation, Copernicuslaan 50, 2018 Antwerpen, Belgium

Abstract. In the coming years the deployment is expected of intelligent context aware services for home, office and city use cases. New enabling platforms are required for the easy development and management of these complex services. Important for the success of these services will be their ability to adapt to changes in the environment in an autonomous manner. There is an important need for an enabling service dealing with the collection, distribution and exchange of context information. In this paper we present CASP, a Context Aware Service Platform taking care of the aggregation and abstraction of context information. CASP is an open platform since uniform interfaces are defined towards context gathering systems and the intelligent services and standards are used to model the context information. Modern persistence techniques are used for the storage of the context information. Applicability is illustrated with two use cases: a desk sharing office web application and an ad-hoc community service in a city environment. In addition some performance results are presented.

1 Introduction

Intelligent context-aware services are currently getting a lot of interest. They allow our homes, offices and cities to become smart environments where highly adaptive services are dynamically deployed, updated or removed. An example is an intelligent city guide that tracks your location and presents information about the neighbourhood. Only information of interest is shown as the city guide is programmed to take care of your interests and user profile. By taking the time of the day into account, at night, the guide suggests cosy restaurants. Or when it starts raining, indoor places of interest are suggested.

For these services to become a success, new adaptive service environments are needed which facilitate the development, deployment and management of such complex services across heterogeneous networks. An important part of these environments will be an enabling service dealing with the collection, distribution and exchange of context information.

T.Pfeifer, et al. (Eds.): Managing Ubiquitous Communications and Services
Fourth International Workshop, MUCS 2007 Munich, Germany, 25th May 2007
© multicon verlag 2007 - multicon lecture notes No. 4 ISBN 978-3-930736-07-2

In this article CASP is presented, a Context Aware Service Platform taking care of the aggregation and abstraction of context information. Context information is gathered in a knowledge base where it is structured according to standards and offered to services and third parties that need or take benefit of this information.

In the next section related work is discussed. Section 3 presents several context modeling techniques and section 4 presents the overall architecture. More details about the implementation are given in section 5. Two use cases, one in a office environment and another one in a city environment are presented in section 6. Evaluation results are discussed in section 7. Finally, our conclusions are stated in section 8.

2 Related Work

Research in context awareness started about 10 years ago. First, specific applications that were tightly bound to the sensor [1,2] were developed. Application developers had to take care of the context acquisition, which made their applications hard to code, maintain and port. Reuse of sensors by several applications was difficult.

In 2000, Dey introduced the Context Toolkit [3], an object oriented Java framework offering a number of reusable components, which allow the rapid prototyping of sensor based context aware applications. The Context Toolkit however lacks a common context model for exchanging context information. Information was simply modeled as key-value pairs.

Recently, architectures were developed with formal context models based on ontologies. Examples are ACAI [4], CoBrA [5], the architecture of the AMIGO IST project [6] and SOCAM [7]. In [8] an analysis of several existing ubiquitous computing systems is made with respect to the context model.

Our platform belongs to this series of architectures. It was described for the first time in [9]. Since then, it has evolved with as main changes generic interfaces towards the context gathering components and the applications, research on a reusable context model for city services based on CityGML and the use of object databases for automatic and efficient storage of context information.

3 Context Modeling

Context, by definition [3] 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 application themselves. This means context involves relevant information about real world entities that needs to be described in a structured model, one way or another, to facilitate the sharing of the collected information. Several context modeling approaches have been proposed in the past: from simple key-value pairs to ontologies.

3.1 Key-value pairs

Key-value pair modeling is the oldest approach to model context. It's usable for context systems with a fixed terminology where the system already knows how to interpret the information. These attribute based models however only allow for exact string matching and thus lack the ability to semantically match concepts in an application indepedent way.

3.2 Markup languages

Markup language based models, such as RDF [10], CC/PP [11], UAProf [12], CSCP [13], provide a more expressive language to structure context information that may or may not be tailored to a specific application domain, while graphical representations based on UML provide a more developer friendly way to represent context information.

3.3 Ontologies

Ontologies are the current state of the art to model context. They come from the knowledge representation field and are used to structure an established vocabulary of terms and concepts within a specific domain of interest. They do this in a way similar to multiple inheritance hierarchies: expressing knowledge about concepts (classes of subjects) and their attributes, as well as their inter-relationships. Concepts can be distinguished by axioms and definitions stated in a formal language, such as description logics. Ontologies enable knowledge acquisition, sharing, reasoning and reuse by formally specifying the concepts in some area of interest and the relationships among them. As such, ontologies provide semantic uniformity and interchangeability of context information in a heterogeneous setting such as pervasive computing environments.

Although numerous ontologies already exist [14–16], it's often a difficult and complex task to reuse or adapt these ontologies for a specific application domain. As our platform is tailored for services in a city, office and home environment, it needed a standard ontology reusable for many applications. Therefore we've chosen to develop a context model based on CityGML [17]. CityGML is an open data model and XML-based format for the storage and exchange of virtual 3D city models.

4 Architecture

Figure 1 gives an overview of the architecture of our platform. The different layers with their main components are discussed in detail below.

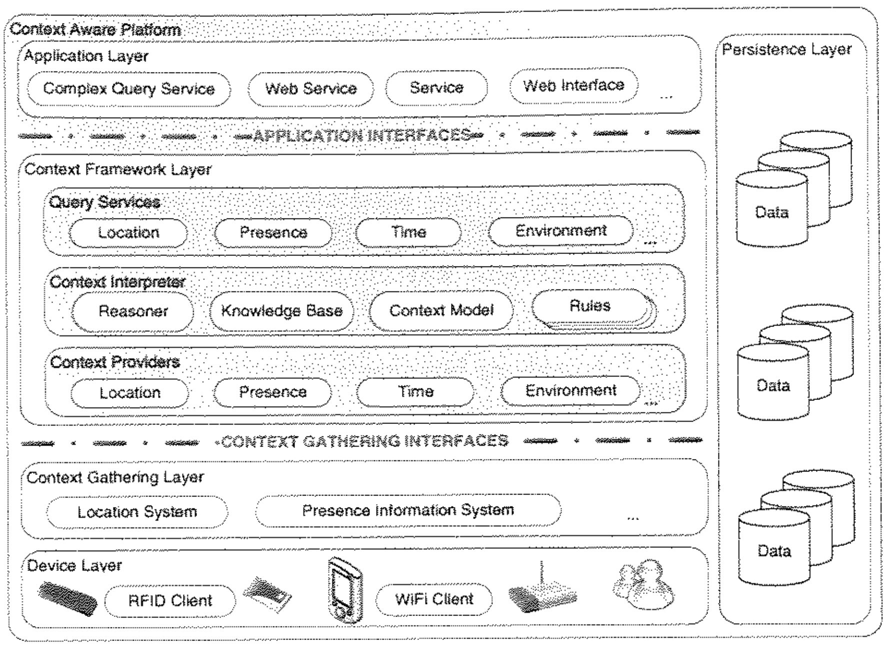


Fig. 1. Overall CASP Architecture

4.1 Discussion of the different layers

Persistence Layer The *persistence layer* ensures persistence of context information. This comprises static context information about users, devices, the environment, etc., but also positions of RFID tags used by context gathering services in the *context gathering layer*. This information is needed when the platform is restarted. It can also be interesting to store dynamic information, like positions of users, together with a timestamp. In this way, the history of information can be exploited and trends or habits can be derived.

The platform supports several database solutions by providing persistency interfaces. This way relational databases like MySQL, postgresql or Oracle or even object oriented databases can be used.

Device Layer The *device layer* includes all devices and software on those devices that deliver context information. For example, a WiFi client on a PDA that measures the signal strengths of access points in the neighbourhood and sends this information to a location system.

Context Gathering Layer The *context gathering layer* takes care of the acquisition of specific context information. For instance location info or presence info. To improve the modularity of the platform, context gathering interfaces are defined between the *context gathering layer* and the *context framework layer* for some important types of context information such as location, presence, time, etc. Implementing such an interface is sufficient for a context gathering component to be pluggable into the platform. Components implementing the same interface can be easily interchanged. As time goes by, new interfaces will be defined for other kinds of context information, since context information can be almost anything.

Context Framework Layer The *context framework layer* is responsible for the aggregation of the context information according to a formal context model and the derivation of implicit information by reasoning. Context information coming from the *context gathering layer* is translated to OWL by the context providers and gathered in a knowledge base. All context providers implement a common interface making them easily pluggable into the platform. As this knowledge base contains all context information, it's the single contact point for services that are in need of context information.

Derivation of extra knowledge is done by providing rules. These are executed by a reasoner using a rule engine providing a forward chaining, a backward chaining and a hybrid execution model. Rules can also be used for validation purposes. For example, if several context providers deliver the same kind of information, there will probably be inconsistencies from time to time. Based on reliability and accuracy parameters, a decision can be made on the correctness of the information. This will be illustrated in section 6.1, where the first use case is presented. The query services enable and facilitate the retrieval of context information. They translate OWL constructs to objects and expose an application interface towards the services. This relieves application developers from writing error-prone queries and translating the results to objects themselves.

Application Layer The *application layer* uses the context information. When a service needs information related to several kinds of context information, a complex query has to be executed on the knowledge base. At that moment, the query services are no longer sufficient. But for a certain application domain, a complex query service can be provided with some frequent occuring queries that can be used by other services, relieving the latter once again from implementing these queries themselves. Communication with other platforms can easily be implemented by providing web services that export the results of the (complex) query services using SOAP.

4.2 Modularity and scalability

Each layer, and therefore the entire platform, is designed in a modular way with a limited number of dependencies. If a context gathering component is stopped or removed from the *context gathering layer*, the context provider in the *context framework layer* will detect this removal and the context information in the knowledge base will be updated. On the other hand, if a context provider is removed from the *context framework layer*, the context gathering components will notice that there is no suitable context provider available and no information will be exchanged. This way, components can be added, started, updated, removed or stopped while the rest of the platform keeps running, the so called white board model [18].

The modular design allows a deployment in a distributed manner. If the load increases, the databases can be deployed on several servers, the context gathering components can be spread in the network and also the knowledge base can be deployed in a distributed manner. This implies tools for automatic duplication of information, information synchronization and distributed queries.

5 Implementation Considerations

In this section the technologies used for the implementation of CASP are discussed in more detail.

5.1 OSGi

The Context Aware Service Platform is currently implemented on an OSGi (Open Services Gateway initiative [19]) service platform, a component-based Java environment which offers standardized ways to manage the software components lifecycles. It's an open platform intended for the delivery and management of services to all types of networked devices in home, vehicle, mobile or other environments. Software components can be installed, updated, or removed on the fly from anywhere in the network. This allows us to add, update or remove context gathering components, context providers or services at any moment.

In addition, the OSGi specifications define a number of standard component interfaces for common functions such as HTTP servers, configuration, logging, security, user administration, XML and UPnP, simplifying development.

5.2 Persistence

As said before, persistency interfaces make the platform independent of the used database technology. In our current implementation, we opted for object oriented databases. More specifically for db4o's native Java open source object database engine [20]. Object oriented databases have some interesting advantages in comparison to relational databases as explained below.

Handling changing data structures A characteristic feature of context information is that it can be anything. This means that, although our context models are based on CityGML, for every application domain and use case, extra information has to be modeled and stored in a database. Moreover the platform is intended for dynamic environments. Constantly, new services are designed and added to the platform. As a result, the context model needs to be adapted or extended during operation. This implies that the data structures will change over time. New data members or new object relationships will be added or even completely new data types representing other kinds of context information. When using a relational database, the schema will likely have to change (to fit the new object structure), and the query code altered to handle the changes. In some cases, this means that a one-time conversion application has to be written to convert the tables to the new format.

Since the context information is represented using objects in an OO language, using an ODBMS instead of a RDBMS means no translation code is necessary to pass data back and forth between row objects fetched from the database and actual objects in your application. Nor is there a need for object/schema mapping code (in case of an object relational database).

Furthermore, the db4o object database that is used, supports native queries [21]. Such queries are not expressed as strings but completely in the implementation language. They are type safe, refactorable and object oriented. So, in contrast to relational queries (SQL), native queries don't have to be rewritten when changes occur. They remain unchanged or are refactored along with the changing object. Automatic schema versioning handles changes in the object model automatically, which renders conversion tools obsolete.

Supporting deep object structures Highly-connected object structures, meaning objects containing references to other objects, possibly containing multiple references themselves, are difficult to store in a relational database. The original structure is often lost in the translation (see figure 2). This makes it hard to maintain the code and keep the stored objects consistent. Ensuring that the "deep" object is stored completely requires large sections of code to be wrapped into transactions.

Most ODBMSes on the other hand implement reachability persistence. This means that any object referenced by a persistent object is also persistent. Typically, the depth to which reachability persistence extends in an object tree, can be specified by the programmer. As a result, a load of objects can be stored or fetched with a single call; the ODBMS engine handles the details of maintaining the references when objects are stored and satisfying them when objects are fetched.

Modeling one-to-many relationships One-to-many relationships require an intermediate table when modeled by a relational database. This table serves as the link between the "parent" object (kept in one table) and the objects in the collection (kept in another table). This is illustrated in figure 2. Trying to store objects of different classes, belonging to one collection, makes this even harder.

Most object databases have no trouble with such an arrangement. The collection is treated as any other object (albeit a potentially 'deep' object, see above) that can be fetched or stored with a single call.

No database administrator (DBA) required In a relational DB, objects have to be "transformed" to store them, meaning that the database has to be designed and effort is needed for an optimal operation of the database. In an object DB, an object can be stored as is. As an object database is used for our platform, no database administrator is required which makes our platform easily deployable.

No performance penalties Database benchmarks [22] show that the db4o object database is an order of magnitude faster than conventional systems. Using an RDBMS requires the overhead of object-relational mapping, resulting in an increased demand on resources.

Moreover, database access by the platform is very low. It is mainly used for platform startup and initialisation of the knowledge base. Queries from services are all targeted towards the knowledge base.

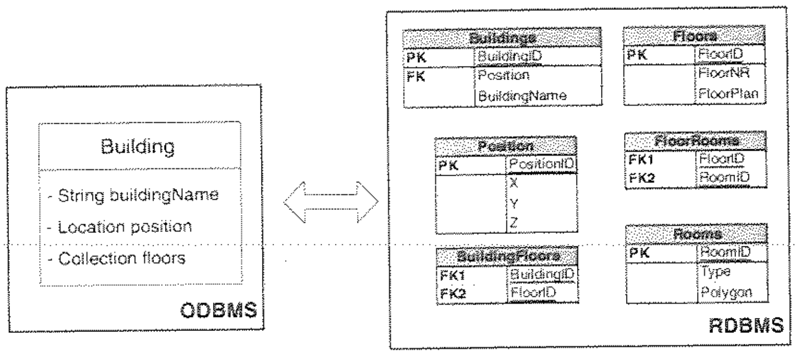


Fig. 2. ODBMS VS RDBMS

5.3 OWL

Our context models are written in OWL (Web Ontology Language [23]), an ontology language proposed by W3C. It's a vocabulary extension of RDF. OWL allows automated processing of terms and the relationships between terms in vocabularies, by representing the meaning of those terms. Domain knowledge can be accurately described by means of classification, modeling dependencies and restrictions on these dependencies. Other ontologies can be imported, encouraging reuse and improving scalability.

5.4 CityGML

CASP is mainly intended for home, office and city services. Therefore, we are currently developing a reusable context model based on CityGML [17]. It is implemented as an application schema for the Geography Markup Language 3 (GML3) [24], the extendible international standard for spatial data exchange issued by the Open Geospatial Consortium (OGC) [25] and the ISO TC211 [26].

CityGML aims to be a generic information model for the representation of all kinds of 3D objects in a city environment that can be shared between different applications. Besides modeling the geometry of these objects, CityGML is able to model appearance and give semantics to different objects in a unified manner.

CityGML enables applications not only to perform visualization, but also extended reasoning and analysis of the modeled environment.

General CityGML supports multi-scale modeling in five so called Levels of Detail (LOD). LOD0 represents the terrain of the modeled environment. LOD1 to LOD4 represent different thematic objects. The higher the LOD, the more details are modeled. In LOD1 buildings are modeled as prismatic blocks. LOD2 adds differentiated roof structures and surfaces. LOD3 models further architectural details of thematic objects. Furthermore, vegetation and transportation objects belong in LOD3. Finally, LOD4 completes the model by adding interior structures such as interior rooms, stairs and furniture.

The base class for every object in CityGML is *CityObject*. It has some general attributes like creation date, termination date and links to generalizations as mentioned above. *CityObjectGroups* can be used to aggregate arbitrary *CityObjects* according to a user-defined criterium. *CityObjectGroups* are themselves *CityObjects*, so nesting of arbitrary depth is possible. A *CityModel*, although also a group of *CityObjects*, is not a *CityObject* but represents the model at hand.

To enable interoperability, restrictions are imposed on attributes which classify objects. An example is the specific type of an object. Rather than using human readable strings which are error-prone, *CodeLists* are used to specify all possible values of the attribute.

It is often useful to have additional information about an object located in an external database. For example information about the owner of a piece of land can be found in a cadastral information system. To model this, each *CityObject* can have *External References* to corresponding objects in external data sets.

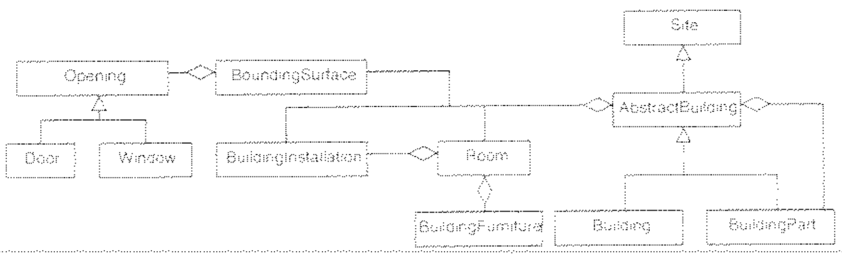


Fig. 3. UML diagram of CityGML's Building model

Building model The most important concept in a city environment is a building. The representation of buildings varies from LOD1 to LOD4. Figure 3 illustrates the building model used in CityGML. Buildings are modeled more or less in the same way as *CityObjectGroups* in that *AbstractBuilding*, the CityGML object used to model a building, can contain a number of *BuildingParts* which are actually just *AbstractBuildings*. In this way buildings of arbitrary complexity can be modeled. *Buildings* can contain *BuildingInstallations*, *BoundingSurfaces* and *Rooms*. In LOD1 a building is modeled by a geometric representation of the building volume.

LOD2 adds the outer facade of the building by allowing *BuildingInstallations* and *BoundingSurfaces* to be part of a building. *BuildingInstallations* represent objects that strongly affect the outer appearance of the building, like balconies, chimneys and such. *BoundingSurfaces* model parts of the exterior shell.

Doors and windows are modeled as *Openings* in *BoundarySurfaces*. This representation, LOD3, is a fine-grained model of the exterior shell of the building.

In LOD4, the interior of the building is represented by the class *Room*. Movable objects in a room are modeled with *BuildingFurniture*, in contrast, objects that are permanently connected to a room such as stairs or pillars are modeled with interior *BuildingInstallations*.

City model Although buildings are an important part of a city environment, there are other things to take into account when creating a virtual city. Objects like lanterns, traffic lights, bus stops, benches can't be modeled as a building. These immovable objects are represented by *CityFurniture*, a subclass of *CityObject*.

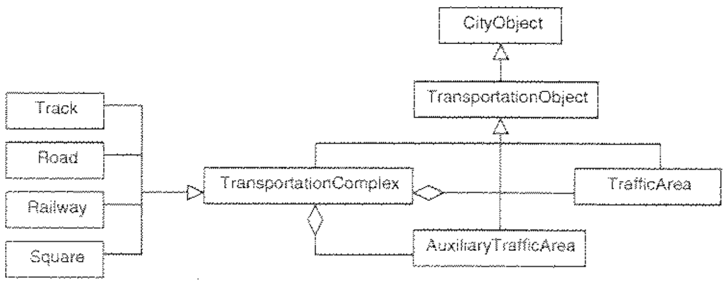


Fig. 4. UML diagram of CityGML's Transportation model

Furthermore, there must be a way for a person, even though virtual, to move from one place to another. CityGML's transportation model provides a manner to represent the possibilities to navigate through the virtual city environment. Figure 4 shows the transportation model. This model is relatively simple. The main class is *TransportationComplex* which is composed of *TrafficAreas* and *AuxiliaryTrafficAreas*. *TrafficAreas* represent the parts of the *TransportationComplex* which are important in terms of traffic usage. Examples of *TrafficAreas* are road lanes, pedestrian zones, parking lots and sidewalks. *AuxiliaryTrafficAreas* on the other hand describe more decorative elements of the *TransportationComplex*.

5.5 Jena2

For the implementation of the knowledge base and reasoner the Jena2 Semantic Web Toolkit [27] was used. This Java library offers an OWL API and a rule-based inference engine.

5.6 SPARQL

Queries performed on the knowledge base by the query services are written in SPARQL, a protocol and query language for RDF. It's a Semantic Web candidate since 2006. It consists of the syntax and semantics for asking and answering queries against RDF graphs. Both the query language [28] and protocol [29] are currently in progress of standardization. SPARQL contains capabilities for querying by triple patterns, conjunctions, disjunctions and optional patterns. It also supports constraining queries by source RDF graph and extensible value testing. Results of SPARQL queries can be ordered, limited and offset in number, and presented in several different forms.

6 Description of Use Cases

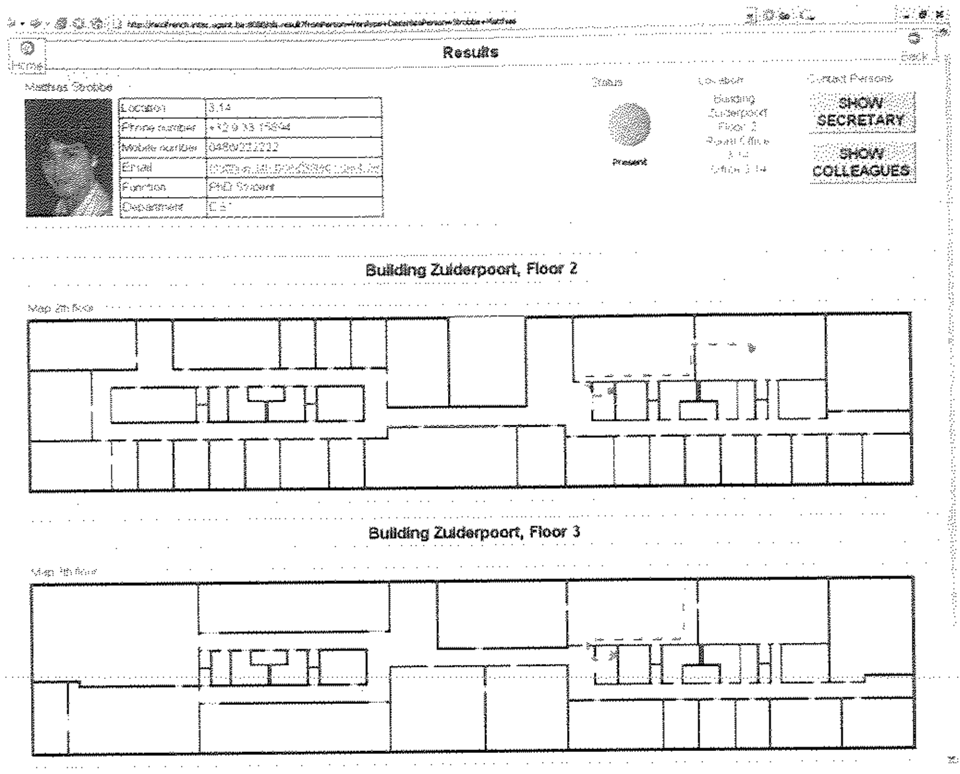
6.1 Desk Sharing Office Use Case

Motivation This first use case is intended for companies with desk sharing. In such companies, employees do not have a fixed location anymore. When they enter the floor or building, they choose an available desk to sit down and start working. In this scenario, several problems can arise. Visitors who have an appointment with an employee do not know where to look. Even an employee who needs another employee for a meeting is not always able to locate the other one.

The desk sharing office use case is a web application consisting of several services. It's main function is to allow employees or visitors to locate other users and to see how someone can be reached. A user has to fill in the name of the person he needs, and as a result the path from his or her own location to the location of that person is shown, together with the status of that person and some extra information like email, phone, etc. Figure 5 gives an example. To realize this, the web interface uses a complex query service to get user, location and presence information from the knowledge base.

When the person that is searched for is not present, a default location can be returned, e.g. the reception or a colleague who might be able to help. It's also possible to see the status of the colleagues, to see if one of them is present. By clicking on an icon, the person can also be called through VoIP.

The application is also useful for management purposes. It allows to trace unknown devices connected to the office network. This is illustrated in figure 6.



Location	3.14
Phone number	+32 9 33 1[illegible]
Mobile number	0486/222222
Email	[illegible]
Function	PhD Student
Department	[illegible]

Fig. 5. Screenshot of the web interface showing the path to an employee

Stars represent known devices belonging to employees. Red dots however are devices that are not known to the system and possibly belong to people that should have no access to the company network.

Ontology For the different services offered by the web application, an ontology for an office environment has been designed. This ontology is shown in figure 7.

A *Person* has a lot of properties containing mostly static information relevant to his job, like the *Departement* he belongs to, his *email* address or his *function*. A *Person* also has one or more *Personal Devices*. Examples are a laptop, a PDA or a mobile phone. These devices can be tracked and thus give an indication of the location of a *Person*. As a *Person* can have more than one *Personal Device* he can have several locations. The determination of his most probable location is done by rules, as will be explained in section 6.1. Presence information is modeled by means of the *status* property of a *Person*. Apart from persons, the office building, the current time and office devices like printers or projectors are modeled.

In contrast to the second use case, this ontology was designed from scratch, so a *Location* is not yet modeled by means of CityGML.

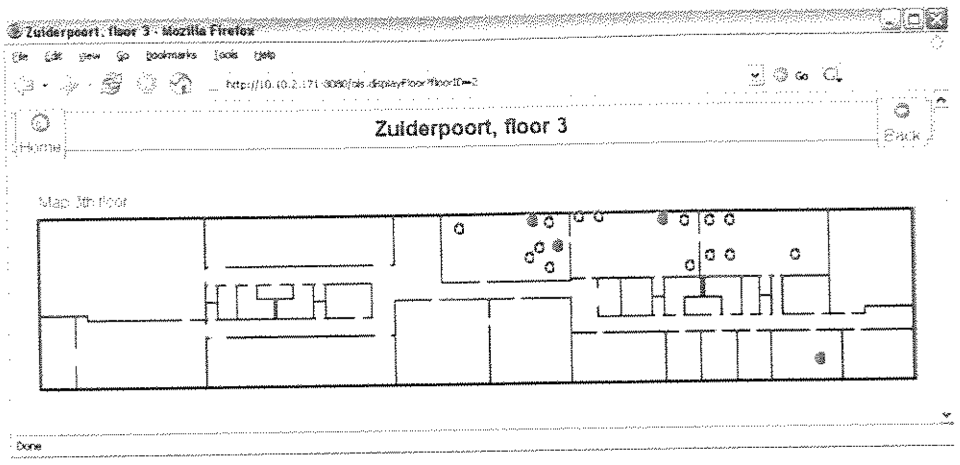

Zuiderpoort, floor 3 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Home
Zuiderpoort, floor 3
Back
Map 3th floor
Done

Fig. 6. Screenshot of the webpage for tracing unknown devices

Location Determination To determine the location of a user, or more specifically the location of a personal device of that user, wired and wireless techniques were implemented. Portables can be tracked by the network cables that connect them with the company network. Such a network cable connects the portable with a certain port of a switch. Each port corresponds with a specific location. Via SNMP the mapping from this port number to the MAC address of the connected device can be queried. Thus, by combining the mappings from the MAC address of the portable to the switch port number and this port to a certain location, the location of the portable can be determined.

Nowadays company networks often also have a wireless (WiFi) part, with access points spread over the building. This allows to determine the location of a user walking around with a WiFi enabled PDA. Therefore the signal strengths originating from the access points have to be measured and compared with a radio propagation model or a radio map [30]. The latter is a database containing measured signal strengths during an offline phase all over the building. An online measurement is compared with these stored values and the best match determines the most probable location. Just considering the measured signal strength as an indication for the distance to a certain access point is not accurate at all inside a building, due to multipath fading and reflections.

In our implementation of wireless location determination, we also use a user profile. Such a user profile calculates the probability that a certain move from one position to another one is made, taking the previous position, the current movement direction and the current speed into account. This improves accuracy as bad measurements, resulting in a location way off the actual location, are filtered out.

Rule Engine When a user has several personal devices (portable, PDA, mobile phone, etc.) and/or one of his devices has several interfaces (wired ethernet, WiFi, etc.), a user can be tracked in several ways. However these locations won't

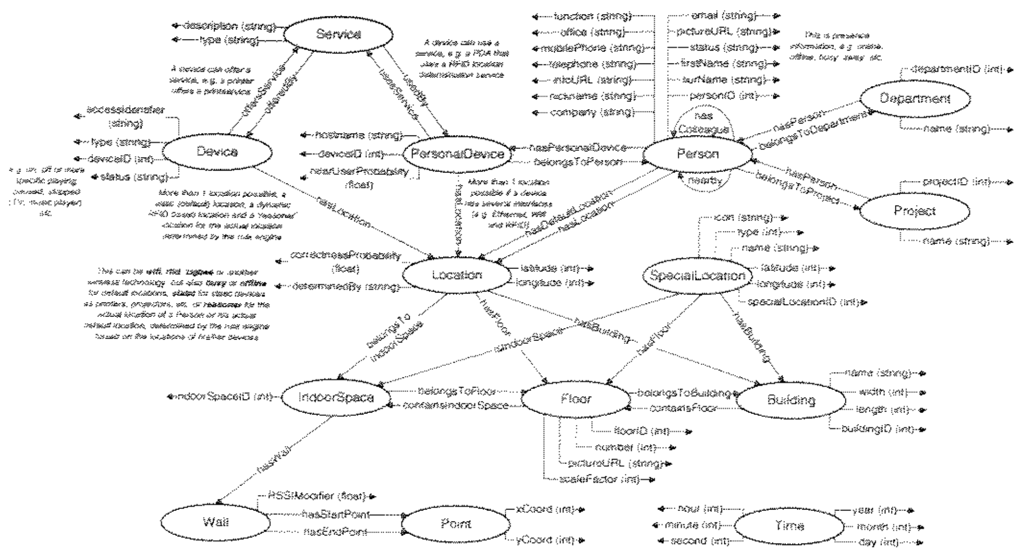


Fig. 7. Ontology for an office environment

be identical at every moment. Think of an employee walking around with his PDA, while his laptop is still on his desk. Moreover, wireless location determination techniques are not perfectly accurate. To determine the most probable location of a user, some extra properties were added to the ontology. A *nearUserProbability* property for a device indicates the chance that a device is in the vicinity of a user. This chance will typically be higher for a PDA than for a portable and even higher for a mobile phone. Furthermore a *correctnessProbability* property was added to every location entity, indicating the accuracy of the location determination technique. The rules we defined take the values associated with these properties into account and choose the location with the highest *correctnessProbability* of the device with the highest *nearUserProbability*.

As an illustration, the following rule adapts the location of a user when a device with a higher *nearUserProbability* is detected.

```
[update_location_person_correctness_probability:
(?x rdf:type ols:Person)
(?x ols:hasLocation ?loc1)
(?x ols:hasPersonalDevice ?y)
(?y ols:hasLocation ?loc1)
(?y ols:nearUserProbability ?nup1)
(?x ols:hasPersonalDevice ?z)
(?z ols:hasLocation ?loc2)
(?z ols:nearUserProbability ?nup2)
greaterThan(?nup2 ?nup1)
-> setLocation(?x ?loc2)
removeLocation(?x ?loc1))]
```

Rules are also used for presence information. When a user has a location, his status becomes *online*. When he leaves the building, his status is automatically changed to *offline*. This last rule is illustrated below. When a user manually changes his location to *busy* or *away*, the rules will assign the user a specific default location associated with the status. Examples are a colleague or the reception.

```
[offline_status:
(?x rdf:type ols:Person)
(?x ols:hasPersonalDevice ?y)
(?x ols:hasLocation ?loc)
(?y ols:hasLocation ?loc)
noValue(?loc ols:hasFloor) // Location entity doesn't exist anymore
-> (removeLocation(?x ?loc))
(removeLocation(?y ?loc))]
```

6.2 Conference Use Case

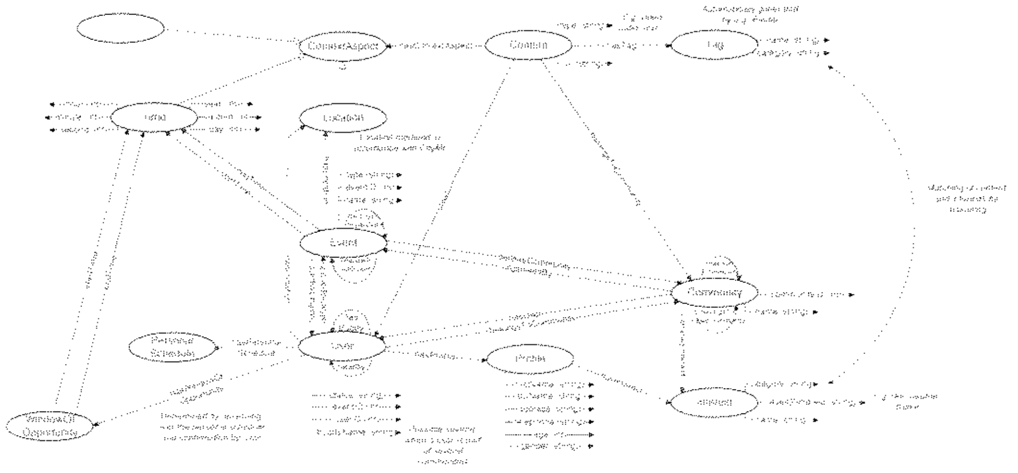


Fig. 8. Ontology for the Conference Use Case

Motivation In a second use case, the CASP platform is used to connect people sharing some kind of interest by means of contextual data.

Consider the following scenario: *John, a Belgian researcher is visiting a conference abroad and, due to his early arrival, he has the time to pay a visit to the city he's staying in. Other persons joining the same social event (the conference) might do the same thing simultaneously, without John knowing this and vice versa. Assisted by his Personal Digital Assistant (PDA) or Smartphone,* *John is able to request (tourist) information to help him explore his temporary unknown environment.*

In this use case, we provide a service towards the end-user that enables him to get to know fellow invitees (of the conference) exploring the city, sharing the same interests. Interests could be subdivided into many different branches. For the purpose of this use case we specifically consider interests to be related to what one would consider interesting while visiting a foreign city, such as: finding information about museums, getting to know the history of the city, looking for some facilities related to sports, specific shops such as a gift shop or a shop with internet connectivity or just looking for a nice restaurant, a cosy bar or the nearest beach.

To achieve this goal, a profile should be built up for each user of the service, representing his interests. Location information is periodically provided by the portable devices. This information, combined with other contextual factors such as the current time and influencing factors from the environment, are all gathered into the CASP knowledge base.

A matchmaking algorithm determines the closest matches to John's profile and then suggests him to send a message to one of those persons. Initially the meeting between John and his new buddy is purely virtual, but if both parties agree they could arrange a physical meeting. The contextual data could be used as an aid to set up a meeting place (e.g. average distance from both current locations and close to a museum).

Ontology The ontology we designed for this use case is presented in Figure 8. A *User* has a *Profile*, which is linked to *Interests*. A *User* participates in some kind of *Event*, such as the conference, which is organised by a *Community*, to which the *User* belongs. Location information is provided by means of a *Location* entity, which is modeled by means of CityGML (see section 5.4). A *User* can see *Content* on his PDA, for example a movie about a historical building. The *Content* that is shown or suggested is dependent on the context of the *User*, the *Time* of the day, his *Location*, the *Community* he belongs to, his *Interests*, etc.

Location Determination Outdoor the main localization technology is GPS. In the near future PDAs will likely have a GPS chip on board, making them easy trackable. As more and more WiFi access points are deployed by companies, institutes but also by private persons, positioning in a city environment based on WiFi becomes possible too. This is demonstrated in [31]. When a user is inside a building the same techniques can be used as described in section 6.1.

Currently research is carried out to develop a good matchmaking algorithm. Once finished, the use case will be further implemented.

7 Evaluation

The *Desk Sharing Office Use Case* is completely implemented and allowed doing some performance tests. Specifically, response times for the web server and the Web Service were measured and analyzed.

7.1 Web Server Request Throughput

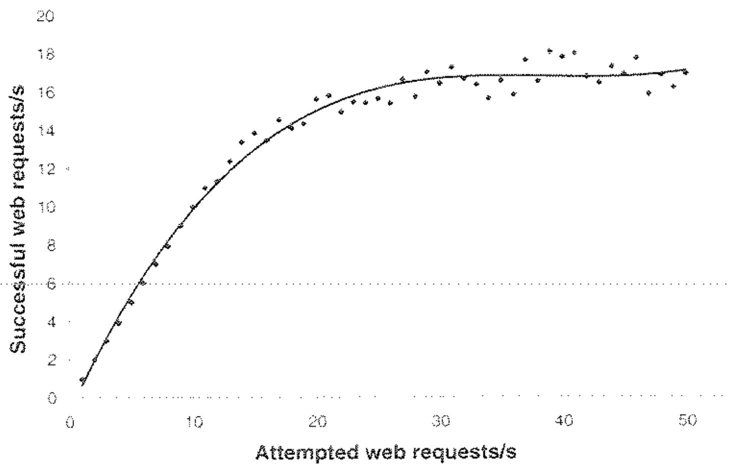


Fig. 9. Attempted Transactions/s vs. Successful transactions/s for the Web Server

To evaluate the performance of the web server, we emulated users using the Avalanche commander equipment, a hard- and software platform developed by Spirent communications [32].

CASP runs on a rack mount PC with an AMD Athlon 3200+ 64-bit processor (2 GHz) and 512 MB RAM. The Avalanche device constantly sends HTTP-requests to the web server. These HTTP-requests consist of five different searches for persons, resulting in web pages as shown on figure 5. To construct this page, the knowledge base is queried for the locations of the requester and the person that is searched for. The shortest path between the two locations is calculated and drawn on a map, along with other information (email, phone, etc.) of the person that is searched for.

As users are connected to a local office network, bandwidth is not a limiting factor. During the tests the number of queries per second varied between 1 and 50.

Figure 9 shows the number of successful requests per second as a function of the number of attempted requests per second. About 17 requests per second can be processed by the server. As a lot of dynamic information has to be requested and calculated, this number is not spectacularly high, but it is sufficient for an average office environment. In comparison, the static home page of the web application can be requested almost 1000 times per second.

As the shortest path between the two persons is calculated (by applying Dijkstra's algorithm [33]) and has to be visualized, performance is dependent on the distance between the two persons. For two persons located next to each other, about 18 successful requests per second were measured, whereas a search between users located on different floors resulted in 11 successful requests per second.

7.2 Web Service Throughput

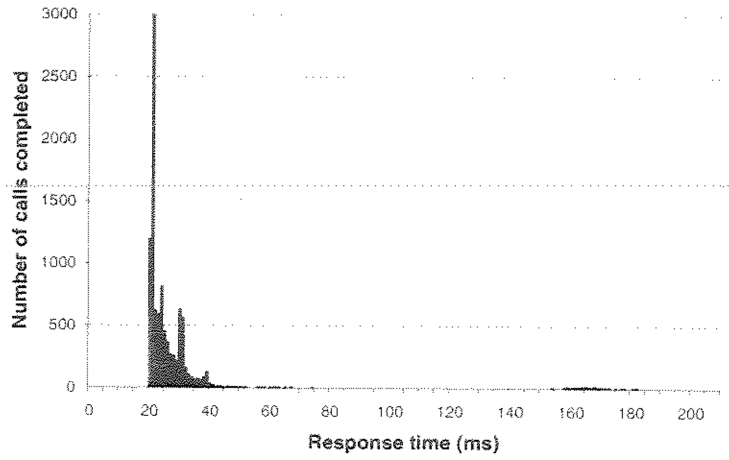


Fig. 10. Response time distribution of the Web Service.

External applications and platforms communicate with the Context Aware Service Platform via the web service API. This API allows third parties to request the context information they need. The response time for getting this information is measured. Figure 10 shows the response time distribution after performing 10000 consecutive calls.

The average response time is 28ms whereas the 50th percentile is 24ms and the 95th percentile 39ms. These are satisfying results, considering that no optimizations for the parsing of the XML messages were used [34].

8 Conclusion

In this paper we presented CASP, a Context Aware Service Platform. This platform takes care of the aggregation and abstraction of context information. When used inside a service enabling platform, it allows the rapid development and deployment of intelligent context aware services. Context information is modeled according to a formal context model based on OWL, allowing easy sharing and reasoning on context information. By using uniform interfaces and standards, the use of context information is transparent to the application developers. Moreover, for supporting a wide range of intelligent city services, context information is modeled according to CityGML, a standard for 3D objects in city environments.

The possibilities offered by CASP were illustrated with two use cases: a desk sharing office web application for the retrieval of the locations of employees and an ad hoc community service in a city environment.

Acknowledgment

The research presented in this paper is partially funded by the IBBT T-Case project and the IST-FP6 Citizen Media project.

Matthias Strobbe is a research assistant of the Fund for Scientific Research - Flanders (F.W.O.-Vlaanderen). Filip De Turck is a postdoctoral Fellow of the Fund for Scientific Research - Flanders (F.W.O.-Vlaanderen).

References

1. Schilit, B. N., Adams, N. I., Want, R.: Context-aware computing applications. In: Proceedings of the 1st International Workshop on Mobile Computing Systems and Applications. (December 1994)
2. Brown, P. J.: The stick-e document: A framework for creating context-aware applications. In: Proceedings of the Electronic Publishing '96. (September 1996)
3. Dey, A. K.: Providing Architectural Support for Building Context-Aware Applications. PhD thesis, Georgia Institute of Technology (2000)
4. M. Khedr and A. Karmouch: Acai: agent-based context-aware infrastructure for spontaneous applications. Journal of Network and Computer Applications **28**(1) (January 2005) 19 – 44
5. Harry Chen: An Intelligent Broker Architecture for Pervasive Context-Aware Systems. PhD thesis, University of Maryland, Baltimore County (December 2004)
6. AMIGO IST Project. [online] http://www.hitech-projects.com/euprojects/amigo/index.htm.
7. Gu, T., Pung, H.K., Zhang, D.Q.: A service-oriented middleware for building context-aware services. Journal of Network and Computer Applications (JNCA) **28(1)** (2005) 1–18
8. Michael Blackstock, Rodger Lea and Charles Krasic: Toward wide area interaction with ubiquitous computing environments. In: Proceedings of EuroSSC 2006. (October 2006)
9. Strobbe, M., De Jans, G., Hollez, J., Goeminne, N., Dhoedt, B., De Turck, F., Demeester, P., Pollet, T., Janssens, N.: Design of an open context-aware platform enabling desk sharing office services. In: Proceedings (on CD-ROM) of PSC2006, the 2006 International Conference on Pervasive Systems & Computing (part of the 2006 World Congress in Computer Science). (June 2006)
10. Dave Beckett: Rdf/xml syntax specification (revised). http://www.w3.org/TR/rdf-syntax-grammar/ (2003)

11. Graham Klyne, Franklin Reynolds, Chris Woodrow, Hidetaka Ohto, Johan Hjelm, Mark H. Butler and Luu Tran: Composite capability/preference profiles (cc/pp): Structure and vocabularies 1.0. http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-2040115/ (2004)
12. WAP Forum: Uaprof user agent profiling specification (1999, amended 2001)
13. Sven Buchholz, Thomas Hamann and Gerald Hubsch: Comprehensive structured context profiles (cscp): Design and experiences. In: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops. (March 2004)
14. Cycorp: Opencyc 1.0: Formalized common knowledge. http://www.opencyc.org (August 2006)
15. Xiao Hang Wang, Da Qing Zhang, Tao Gu and Hung Keng Pung: Ontology Based Context Modeling and Reasoning using OWL. In: PERCOMW '04: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops. (2004)
16. H. Chen, F. Perich, T. Finin and A. Joshi: Soupa: Standard ontology for ubiquitous and pervasive applications. Mobile and Ubiquitous Systems: Networking and Services (2004)
17. CityGML: http://www.citygml.org/.
18. OSGI Alliance: Listeners considered harmful: The "whiteboard" model. Technical Whitepaper
19. The Open Services Gateway Initiative (OSGi). [online] http://www.osgi.org.
20. db4objects: http://www.db4o.com/.
21. William R. Cook, Carl Rosenberger: Native queries for persistent objects a design white paper. [online] (August 2005) http://www.db4o.com/about/productinformation/whitepapers/.
22. db4objects: benchmarks. [online] http://www.db4o.com/about/productinformation/benchmarks/.
23. W3C: OWL Web Ontology Language Overview. [online] http://www.w3.org/TR/owl-features.
24. Geography Markup Language 3 (GML3). [online] http://www.opengeospatial.org/standards/gml.
25. Open Geospatial Consortium (OGC). [online] http://www.opengeospatial.org.
26. ISO/TC 211. http://www.isotc211.org
27. Jena 2 Semantic Web Toolkit. [online] http://jena.sourceforge.net/.
28. W3C: SPARQL Query Language for RDF. [online] (2006) http://www.w3.org/TR/rdf-sparql-query.
29. W3C: SPARQL Protocol for RDF. [online] (2006) http://www.w3.org/TR/rdf-sparql-protocol.
30. Youssef, M., Agrawala, A.: On the optimality of wlan location determination systems. In: Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference. (January 2004)
31. Cheng, Y., Chawathe, Y., LaMarca, A., Krumm, J.: Accuracy characterization for metropolitan-scale wi-fi localization. In: Proceedings of Mobisys 2005. (2005)
32. Spirentcom Avalanche. [online] http://www.spirentcom.com/avalanche/.
33. E. W. Dijkstra: A note on two problems in connexion with graphs. Numerische Mathematik 1 (1959) 269–271
34. Kostoulas, M.G., Matsa, M., Mendelsohn, N., Perkins, E., Heifets, A., Mercaldi, M.: Xml screamer: an integrated approach to high performance xml parsing, validation and deserialization. In: WWW '06: Proceedings of the 15th international conference on World Wide Web, New York, NY, USA, ACM Press (2006) 93–102

In Ubiquitous and Pervasive Communications novel information and communications management technology are required to support dynamic, integrated management of participants, information appliances and smart space infrastructure.

Ubiquitous Communications, as evidenced in pervasive computing and smart space applications, still present significant management challenges for the successful delivery of highly adaptive services across heterogeneous networks, middleware, applications and devices. Today's management systems are still unable to cope with the complexity, heterogeneity and automation required by the pervasive computing vision. New paradigms, models and technology need to be developed to allow computer systems manage themselves in accordance with high-level guidance from humans. This workshop proceedings explore the theoretic, technological and organisational challenges in managing ubiquitous communications and application services.

The Fourth International Workshop on Managing Ubiquitous Communications and Services (MUCS 2007) was held in Munich, Germany, 25th May 2007, as part of the IEEE Integrated Management conference, IM 2007.

ISBN 3-930736-07-1

9 783930 736072 >

Tom Pfeifer

John Strassner

Simon Dobson (Eds.)

Managing
Ubiquitous Communications and Services

Fourth International Workshop, MUCS 2007

Munich, Germany, 25th May 2007

Proceedings

multicon lecture notes – No. 4

multicon verlag

Schöneiche bei Berlin