VII Simpósio Brasileiro de Geoinformática, Campos do Jordão, Brasil, 20-23 novembro 2005, INPE, p. 15-29.

Using Service-Oriented Architecture in Context-Aware Applications

Damião R. Almeida, Cláudio S. Baptista, Elvis R. da Silva, Cláudio E. C. Campelo, Camilo P. Nunes, Bruno A. D. da Costa, Wilkerson de L. Andrade, Jairson M. Cabral

Computer Science Department – Universidade Federal de Campina Grande (UFCG)
Rua Aprígio Veloso, 882, 58107-970, Campina Grande - PB, Brazil

{damiao,baptista,elvis,campelo,nunes,bcosta,wilker,jairson}
@dsc.ufcg.edu.br

Abstract. *The enhancements in mobile device infrastructure enable the use of a location computer service while user is on the move; such systems are known as Location-Based Services (LBS). When these applications manage user profile and context, they are coined context-aware applications. In this paper we propose a service-oriented architecture for context-aware applications named Omnipresent. Omnipresent is based on well-established standards such as Web services, and those from the OpenGeoSpatial Consortium. Omnipresent offers several services including map presentation, routing, advertisement, and also works as a reminder tool. Users may use Omnipresent either by mobile devices or Personal Computers.*

1. Introduction

The enhancements in mobile devices which include better processing power and large storage capability have motivated the development of more sophisticated embedded applications. Nowadays such mobile devices enable the use of a remote computer service while user is on the move. This gives raise to a new computer paradigm known as ubiquitous computing, which provides computing services anywhere and anytime [Feng et al. 2004].

In the approaching Information Society, people will be surrounded by electronic sensors and they will use a set of environment states, which guide application behavior according to a context. These applications are known as context-aware [Chen and Kotz 2000]. Context-aware applications are sensible to user necessities, personalized according to its profile, requirements and context. They pro-actively help users in daily activities through user-friendly interfaces.

Context has been categorized in user-centric and environmental. According to Apers [Feng et al. 2004], user-centric context may be classified into:

- Background: information related to user profile, its opinions and working place;
- Dynamic Behavior: information related to user habits, its tasks and daily activities;
- Physiological State: information obtained from sensors such as temperature, blood pressure, heart beat, etc;

• Emotional State: information obtained through visual and acoustic analysis, or directly provided by the user.

Yet, environmental context is classified as:

• Physical Environment: information about the environment. For instance, location, temperature, weather, noise level, lightness, etc;

• Social Environment: information about people nearby, shop offers, traffic jam, etc;

• Computational Environment: information about electronic devices available to the user.

Location-Based Services (LBS) may integrate information about geographic location and context. This work presents a service-oriented context-aware LBS architecture, known as Omnipresent, based on push and pull services [Schiller and Voisard 2004]. In push mode, user does not control the time when information is going to be delivered to himself, hence maps with Points of Interest (PoI) and other information may be sent automatically to the user, according to his profile and context. The aim of our system is to help user in finding products, services, PoIs, friends nearby, and the management of his daily tasks according to context.

The proposed service-oriented architecture is based on Web Services technology [Alonso et al. 2004]. Thus, there exists a LBS service which interacts with other services such as map service, routing service, advertisement service, through a standard WSDL interface. These services may be published in a UDDI (Universal Description, Discovery and Integration) directory.

The remainder of this paper is structured as follows. Section 2 presents context modeling. Section 3 focuses on the overall architecture. Section 4 presents a prototype using the proposed architecture. Section 5 discusses related work. Finally, section 6 concludes the paper and highlights further work to be undertaken.

2. Context Modeling

Our context model is based on ontologies using OWL (Ontology Web Language) [W3C OWL 2005]. Ontologies are useful for context modeling as they map the three basic concepts in a context model:

• Class: which represents existing things in a domain, such as user, profile, products, appointments, etc;

• Relationship among classes, for instance, located_at, likes, works, etc;

• Class properties or attributes, e.g. user name, date of birth, blood pressure, temperature, etc.

Also, ontologies may use rules for context reasoning. For instance, rules can be expressed using the Jena framework by using pre-conditions and conclusions such as:

[rule1: (?f omnipresent:father ?a) (?u omnipresent:brother ?f) -> (?u omnipresent:uncle ?a)]

Figure 1 presents part our ontology for context modeling. The *User* class is the main concept. The ontology may be divided into context categories such as:

- *Background* is represented by the *Profile class* and its subclasses. User profile is classified into three categories: personal, professional, and social profiles. Personal profile contains information about user interests, marital status, date of birth, etc. Professional profile contains information about related job activities such as job title, email, company name, and so on. Social profile contains information about social behavior such as religion, smoking and drinking customs;

- *Dynamic Behavior* is represented by the *Task* class which contains user tasks and meetings. The *Task* class may contain either the interest on buying or selling products; or a scheduled meeting. Either products or services may be advertised. Both people and shops may advertise products or services. This advertisement is going to be pushed to clients whenever they are close to the shop or seller, and are obviously interested in the product or service. In Figure 1, the *Appointment* class represents a scheduled appointment. However, these appointments are reminded not only by using a specific date and time, but also according to user context. For instance, the system is able to trigger a reminder for "remind me to call a friend when both of us are idle and his emotion status is happy";

- *Physiological State* is represented by the *Physiologic* class which contains information about client physiologic state such as: heart beat, temperature, blood pressure, etc;

- *Emotional State* is represented by the *Emotion* class which indicates user emotional status: sad, happy, nervous, bad-humor, etc.

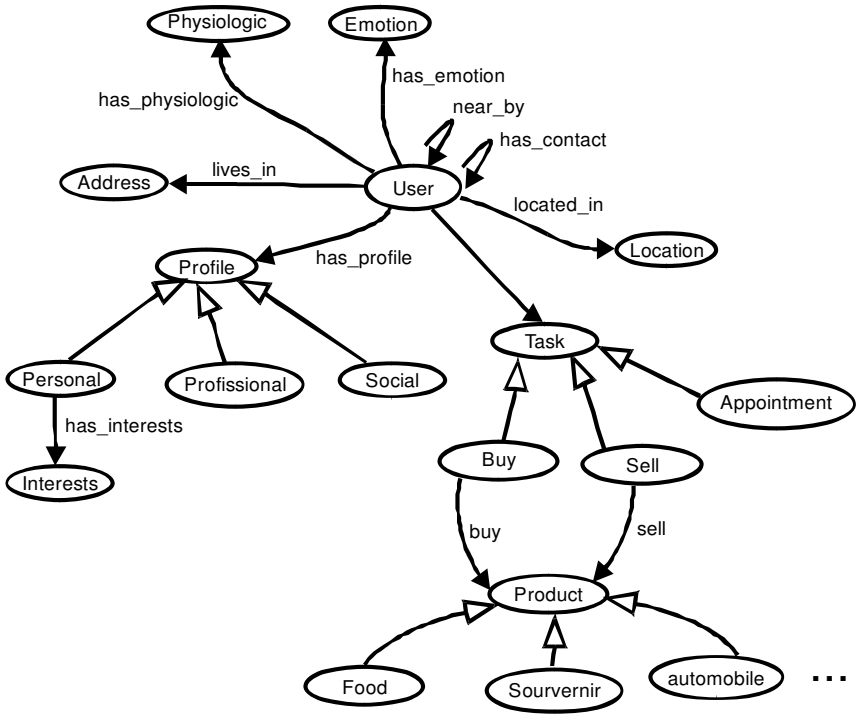

Figure 1. User context ontology

2.1. Action triggered by context

As in the OSGi framework [Gu et al. 2004], some actions may be triggered according to context changing. Users may define decision rules and specify the action to be triggered (e.g. send an email, activates an emergency service, or send a specific message to client). These rules are deployed using the Jena framework, which enables inference. Table 1 presents some of such context based rules.

Table 1. Rule-based actions

Remind	Rule
Whenever my son leaves school during class time, I would like to be notified in my mobile device.	If (User's son ¬located_in school) Λ (User's son's task status busy) then active message ("Son out of the school")
Remind me to pay the bills when I am close to a bank.	If (User near_by_location bank) Λ (bank status open) then activate reminder ("pay the bills")
Remind me to call to my friend John when both of us are idle and he is not angry.	If (task status iddle) Λ (John's task status iddle) Λ (John's humor ¬angry) then activate reminder ("call john")
Call an emergency service whenever a sick man´s overall health status is bad	If (heart_beat > threshold_1) V (blood_pressure > threshold_2) V (temperature > 101 ºF) then activate alert ("Call 190")

3. The Omnipresent Architecture

The Omnipresent service-oriented architecture is based on four services: a LBS Web service, a Presentation Web service, a Routing Web service, and an Advertisement Web service. Moreover the system still provides a client application which is responsible for user-interaction with these services. By using such application, clients may fulfill forms which enable the system to obtain information about their lives, appointments, and daily activities. This information is collected and sent to the LBS Web service.

Web services encapsulate the underlying applications and publish them as a service which may be accessed remotely and enables reuse of existing services using a set of standard protocols and formats. By using Web services, the load balancing may be well distributed so that scalability and better performance on query execution may be achieved [Nagappan et al. 2003]. Another advantage of using Web services is related to the rapid application development and reuse of existing components.

In the following, the Omnipresent services are described in more detail. Each service has a correspondent WSDL file which describes its interface.

3.1. The LBS Web Service

The LBS Web service is the main service in our architecture. It is responsible for receiving and managing user context information. When a client is registered and provides profile, advertisements, and appointments, it is able to receive context-aware information. The main available operations are:

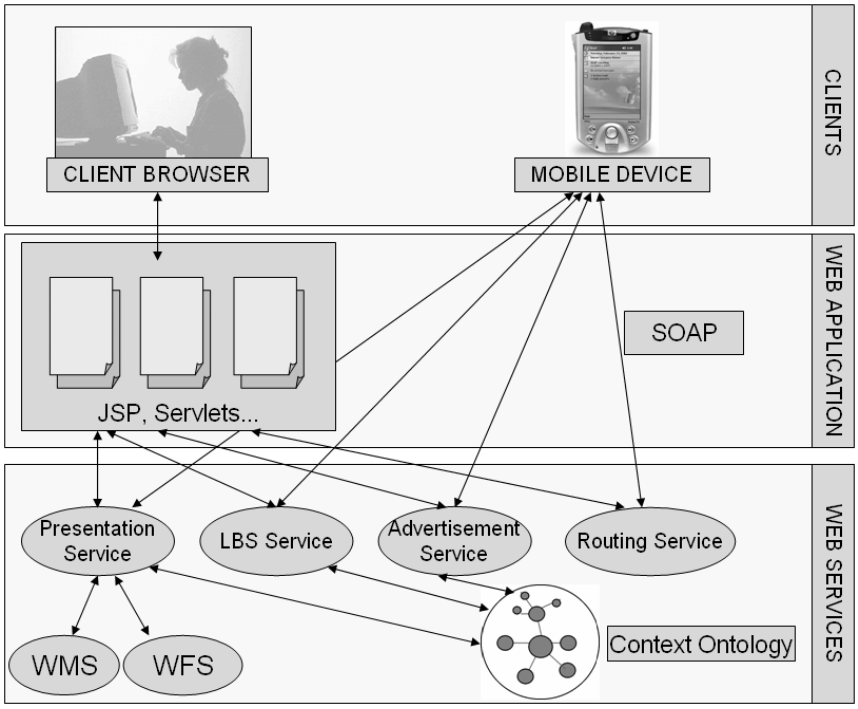

Figure 2. The Omnipresent architecture

• registerUser() which registers a user in a Web service, its profile and context information;

• registerCompany() which registers a company interested in publishing products or services;

• updatePosition() which enables to update user geographic location in the system, and analyses the context searching for actions which are relevant to user context. Each time user location is updated the context is analyzed;

• updateEmotion() which updates user emotional status;

• updatePhysiologic() which updates physiological status;

• registerAppointment() which registers a user appointment.

3.2. The Advertisement Web Service

This Web service manages products and services offered by users. Products are organized according to their ontology description. New categories and services may be added to this service. This service is based on the Directory Service of the OpenLS Service specification [OpenGeoSpatial LS 2005]. However, this service extends that specification by providing a semi-structured way for describing products, so that better retrieval can be achieved. We use a XMLSchema to describe products and services, and the existing categories may be expanded whenever necessary. For example, let us suppose that a car is a new product to be registered in the system. Therefore, it is necessary to provide a XMLSchema which describes this product, so that, each instance of this category will have a well-defined structure. Figure 3 presents part of a

XMLSchema which describes the features of a car product. In this example a car has the following properties: make, year, miles, number of doors, color, and fuel.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
    <xs:element name="car">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Make" type="xs:string">
                <xs:element name="Year" type="xs:integer">
                <xs:element name="Miles" type="xs:float">
                <xs:element name="Doors" type="xs:integer">
                <xs:element name="Color" type="xs:string">
                <xs:element name="Fuel">
                    <xs:complexType>
                        <xs:choice>
                            <xs:element name="alcohol" type="xs:string">
                            <xs:element name="diesel" type="xs:string">
                            <xs:element name="gas" type="xs:string">
                        </xs:choice>
                    </xs:complexType>
                </xs:element>
            …
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figure 3. An example of an XMLSchema for a product

Hence, by using such XML documents, a Web application may discover product properties. In Omnipresent, a Web application searches for available products and services in the Advertisement Web service. Figure 4 presents a product search form which was built according to the XMLSchema of Figure 3. Users may register a demand for a specific product or service. Then the Advertisement service searches for products demanded by logged users, while the user is on the move.

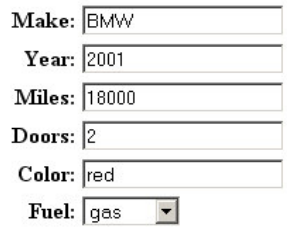

Figure 4. An example of a form generated from the XMLSchema

3.3. The Presentation Service

This Web service is responsible for providing maps of a given geographic area. The Presentation Service interface is based on the OpenGeospatial OpenLS specification [OpenGeoSpatial LS 2005]. Nonetheless, we extend this Web service with the capability of analyzing user profile so that maps are provided with PoIs according to user context. For instance, if a user is visiting a city and likes seeing paintings, the system may locate museums and expositions and send to the user a map of the region containing such PoIs.

In the Presentation Service, the request parameters are specified in the MapRequest class, the design of which was based on the OpenLS Presentation Service specification. According to this specification, the map request is divided in three parts: Output, Basemap and Overlays. In the following the first two will be described, the overlay one is part of the routing service which is going to be detailed in the section 3.4:

- Output – which specifies the map size, format, transparency, content type (e.g. URL o base64 data), etc. The *output* also contains a class called BBoxContext which has two points in a given spatial reference system and the dimension which specifies the bounding box of the area of interest. Only the BBoxContext and the width, height and format attributes are mandatory;
- BaseMap – which specifies the layers which will be used in the map. Each layer is composed of name and style. According to the OpenLS specification, the BaseMap is not mandatory, however in our design we have decided to become the BaseMap mandatory in order to achieve compatibility with the OpenGeospatial Web Map Service specification, as in the latter it is necessary to provide at least one layer.

Figure 5 presents the MapRequest class diagram, using the UML notation.

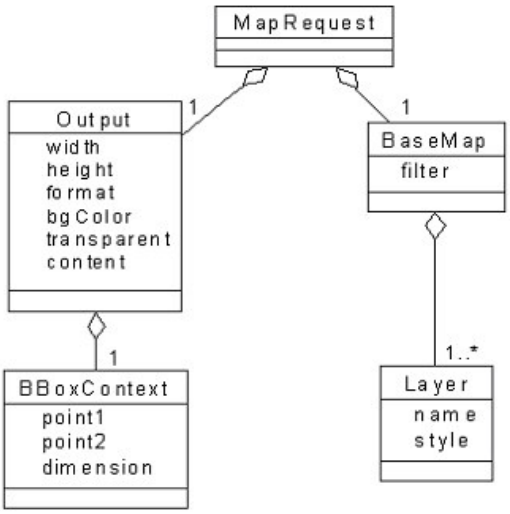

Figura 5: MapRequest class diagram

The Presentation Service works together with the Web Map Service and the Web Feature Service. These services are based on the OpenGeospatial Web Map Service specification – WMS [OpenGeoSpatial WMS 2005] and Web Feature Service specification - WFS [OpenGeoSpatial WFS 2005]. Nonetheless, we have implemented a Web service interface for such services. A request is done through several parameters in a URL. By implementing these services as Web services we not only make the requests easier, but also it is possible to publish these services in a directory services such as UDDI, in order to provide automatic service discovery and invocation [Alonso et al. 2004].

Currently, we have implemented the following WMS operations: *getCapabilites*, *getMap* and *getFeatureInfo*. The *getCapabilites* method obtains service metadata which

describes relevant information including maximum number of layers which may exist in a map, layers and styles available, coordinate reference systems (CRS), bounding box and scale. The *getMap* method returns a map according to the parameters received. These parameters include Layers, which contains the list of layers in a map; CRS, which contains the coordinate reference system; BBOX, which expresses the bounding box of interest; and Format, which contains the map output format. Currently we are working with the SVG format. Lastly, the *getFeatureInfo* method enables to retrieve more information about a chosen feature, for instance, a hotel name, address, number of rooms, etc. The maps provided by the Presentation Service are obtained from the WMS, however it is the former which is responsible for map personalization using the user PoI.

The Web Feature Service specification enables the user to query and update geospatial data is a interoperable way. This service contains the following operations: *getCapabilities*, *describeFeatureType* and *getFeature*. The *getCapabilities* method is responsible for describing available features, and the operations supported by each feature. The *describeFeatureType* method obtains the structure of a given feature type, which is described in a XMLSchema. Finally, the *getFeature* method enables users to specify which feature properties will be queryed, as for example the name of a given feature.

Figure 6 presents the architecture of the map service. There is a unique Web Service interface, which encapsulates the WMS and the WFS web services. All requests are implemented through remote function calls in the Web service.

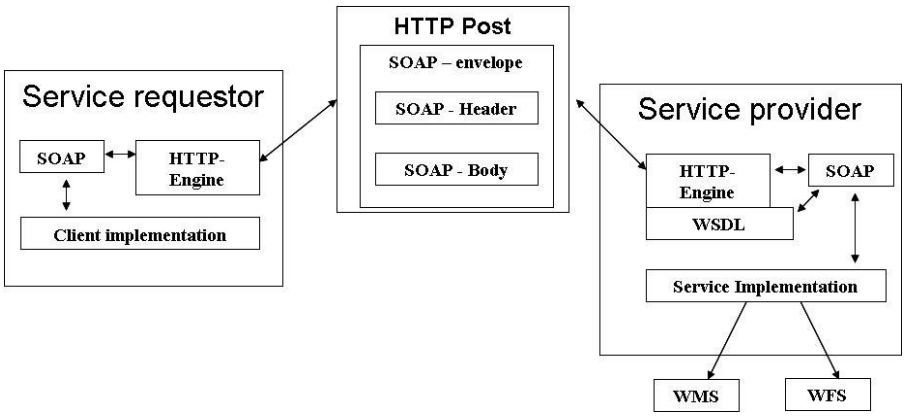

Figure 6. Map service architecture.

3.4. The Routing Service

This service follows the OpenGeoSpatial OpenLS specification. This routing service is responsible for providing on demand routes among two or more places. The client sends a request to this service by providing two points. Besides these two points, the client may inform some preferences to be taken into account by the service such as road and traffic conditions. The service receives the request, and then calculates the route and sends it back as geometry (path) to the client. This geometry may be used by a presentation service in order to render a map with the specific route highlighted.

The routing Web service stores data about roads and their intersections as a graph. The path between each two pairs is pre-computed and stored in the database,

using the total materialization strategy [Shekhar et al. 1997]. The cost of such storage is high, but as disk costs are decreasing, the gains obtained in processing power are worthy. We use a graph hierarchy to minimize the storage costs [Jing et al. 1996]. The graph is partitioned in non-interleaving sub-graphs and each partition is represented by a node in a super-graph. There are nodes which take part in more than one sub-graph. They are called border nodes and take part in the super-graph.

Paths are computed in each fragment and in the super-graph. Hence, best path can be computed according to the following possibilities:

- source and target nodes take part in the super-graph. Hence, it is sufficient to return the pre-computed path between them;

- the source node is not in the super-graph, but only the target one. Thus, an intermediary node in the super-graph should be encountered. This node should be in the same partition of the source node. Thus, the path from the source node to the intermediary one plus the path from the intermediary node to the target one is the full path returned;

- the source node is in the super-graph, but the target one is not. This situation is symmetric to the previous one;

- neither source nor target nodes are in the super-graph, but they are in the same partition. So, it is sufficient to return the pre-computed path between them;

- neither source nor target nodes are in the super-graph, and they are in different partitions. Therefore, a border node in the source node partition and a border node in the target node partition should be encountered. The best path is going to be the union of the best path from the source node to its partition border node, and the best path from the target partition border node to the target node.

We used the classical Dijkstra best path algorithm to pre-calculate all costs [Dijkstra 1959]. This algorithm is simple and once we are maximizing performance in our approach due to pre-materialization of all paths, then it has been used with success. We use weights for computing both road and traffic conditions. Obviously, the path with minimum weight is stored.

Users may request simple shortest path, shortest path using road condition, shortest path using traffic condition, and a mixture of the two latter. When a client requests for a best path, the service accesses the database of materialized paths, searches for it, and returns the geometry to the client. This Web service has a unique method: getRoute(), which receives as parameters the route type (for instance “shortest”), two pairs of latitude and longitude coordinates for the source and target; a Boolean which indicates whether user takes into account traffic condition; and another Boolean which indicates whether user takes into account road condition. The getRoute() method returns the path in SVG and the total distance.

4. Implementation issues

The Web services were implemented using the Java J2EE and JAX-RPC. Each service was developed in three steps: interface definition, which contains the methods which may be called by a client; interface implementation, which contains the service logic;

and the service deployment in an application server. We have used the Java Sun Application Server. After the second step is done, the WSDL file is generated automatically. The WMS, WFS and Routing services were implemented as an extension of the iGIS, which is a framework for rapid application development of Web GIS [Baptista et al. 2005]. The advantage of using such framework is that the maps may be stored in any spatial database system, such as Postgresql, Oracle, IBM DB2 and MySQL. Moreover, this framework provides two map formats: vector (SVG) and raster (JPEG or PNG).

In order to validate the proposed architecture we have built a prototype in a PDA using a PocketPC running Windows CE v3.0 and Jeode Java Virtual Machine. A GPS receiver was connected to the PDA in order to capture location. At the moment, the GPS is the only real sensor we have, so that other information such as physiological and emotional information is simulated.

The first thing the user should do is to register itself in the system and describe its profile through a Web form. This information is sent to the LBS service which stores it. The registration of reminders, appointments, and meetings may be done through Web forms, so that the system is not limited to the usability restrictions of mobile devices.

After registration, the user may interact with the LBS service. An authentication procedure, using login and password, was implemented for security reasons. The LBS service function calls are implemented using the SUN JAX-RPC package. This package is a version of the SUN JAX-RPC for mobile devices with limited capacity [Yuan 2003].

Map rendering in the PDA uses the SVGT (Scalable Vector Graphics Tiny) standard. SGVT is a version of SVG for mobile devices [W3C SVG 2005]. Usually, a map in SVGT has smaller size than the same map in JPEG, GIF or PNG formats. Moreover, SVG provide some spatial operations such as zooming in and out and panning, visible layers, and so on. These operations are processed in the client, without needing to invoke the server.

When the PDA sends the geographic location to the LBS service, the user profile and context are analyzed, and then the PoIs and advertisements of user interest are sent to the device. By monitoring user context, the system may find an appointment or reminder registered that should be alerted. When a message concerning a particular appointment is sent to the client, it is marked in the database as sent. Other information may be obtained such as friends nearby, product advertisement, PoIs, and routing. When the client position is near the map bounding box, another request is sent from the device to the Presentation Service in order to obtain a new map.

Figure 7 shows a form in which user may choose his interests by providing a weight (from none to 5) to each one chosen. This procedure helps to specify user profile, and the system may choose PoIs according to user preferences. Figure 8 presents the interface in the mobile device, which contains a map with hotels. The circle shows a PoI. The user may point to that particular PoI and obtain detailed information, as presented in Figure 9. User may invoke the routing service to obtain a route from his current location to that particular PoI, as shown in Figure 10. The desktop client has

been developed using the iGIS framework. Figure 11 presents a screenshot of the Omnipresent PC client.

Figure 7. Profile definition

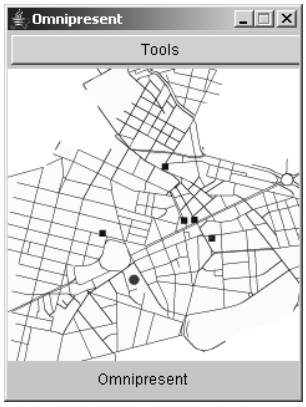

Figure 8. Event of interest represented by a circle

Figure 9. Information about the event

Figure 10. Routing to the PoI

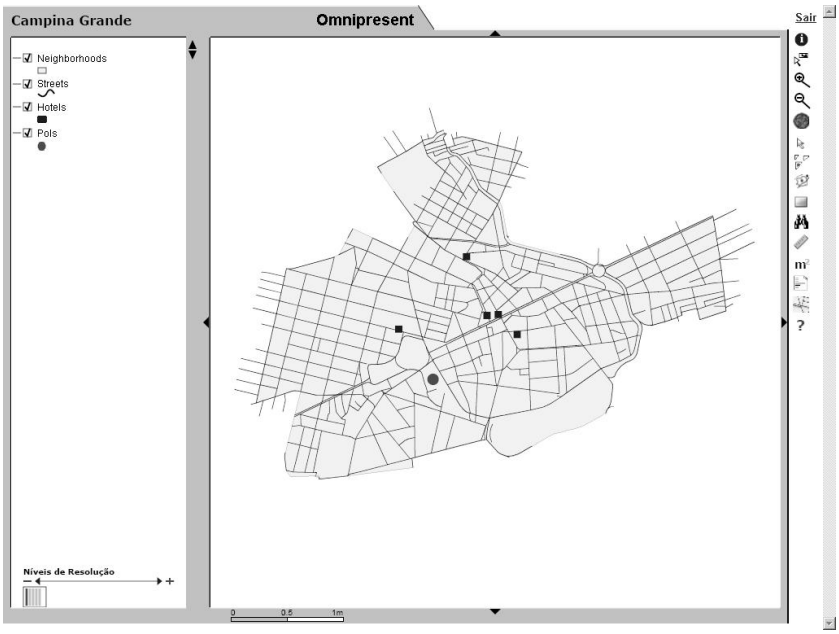

Figure 11. PC Client

5. Related Work

Following the advances in ubiquitous computing, many context-aware tools have been proposed in the literature so far. Some of these tools provide push service, however they do not take into account user context as it is the case of the Online Aalborg Guide [Andersen et al. 2003]. This framework is based on LBS and it was developed aiming to serve as an online guide for tourists who visit the city of Aalborg in Denmark. The main features of this tool include map rendering with information about PoIs close to user location and according to user profile; and shortest path to a given PoI. This system provides both push and pull technologies. A request to the server is done every time user asks for information of a given PoI. One of the main drawbacks of the Aalborg Guide is that the system is limited to the nearest PoIs, and there is no analysis based on context information.

FLAME2008 [Weißenberg el al. 2004] is a context-aware tool which uses user profiles in order to provide relevant services. However the system was designed for a specific domain: the Beijing Olympics Games in 2008. Moreover, information about people nearby, such as friends and relatives, is not provided.

AROUND [Jose et al. 2003] provides a LBS architecture which enables to locate services according to a given spatial area. The main drawback of this tool is the lack of context modeling.

Moreover, there are reminder tools such as ComMotion [Marmasse and Schmandt 2000] and CybreMinder [Dey and Abowd 2000]. A reminder is a kind of special message that is sent to ourselves or to other people, in order to inform about activities that we need to undertake. For instance, "at 10:30 a.m. remind me about a meeting with the boss to discuss a specific problem". The ComMotion tool uses space and time to present the registered reminders. CybreMinder provides a more complex environment for dealing with context, as people nearby and weather condition are taken into account.

Omnipresent works as both a reminder tool, and as a context-aware LBS system. Omnipresent enables personal management with information about user and the surround environment. The tool uses both push and pull technologies. The user may access the software not only via mobile devices but also from any computer connected to the Internet. Moreover, differently from FLAME2008, Omnipresent is domain independent. Lastly, Omnipresent is based on well-established standards such as Web services, OpenGeospatial Web Map Service [OpenGeoSpatial WMS 2005] and OpenLS specifications [OpenGeoSpatial LS 2005].

6. Conclusion

The advances in mobile computing, sensors and wireless networks have motivated the development of context-aware applications. These applications evaluate user environment and pushes information which are relevant to user context. In this paper we present Omnipresent, which is a service-oriented architecture for context-aware. Omnipresent may be accessed from either mobile devices or PC, is based on Web services and well-established standards for LBS applications such as those proposed by the OpenGeoSpatial Consortium. Omnipresent offers several services: map presentation, routing, advertisement, and also works as a reminder tool. Users may also locate friends and search for their context (e.g. emotional status, health status, etc.).

As further work, we intend to use sensors to get more contextual information, more specifically to deal with Physiological environment, as currently we are simulating such information. Also, we intend to implement a client on smart phones. Finally, an evaluation study, which includes usability and overall performance tests, is due.

References

Alonso, G., Casati, F., Kuno, H., Machiraju, V. (2004) "Web Services: Concepts, Architectures and Applications", Springer-Verlag, Berlin Heidelberg New York.

Andersen, K. V. B., Cheng, M., Klitgaard-Nielsen, R. (2003) "Online Aalborg Guide – Development of a Location-Based Service", Technical report, Aalborg Universitet.

Baptista, C.S., Nunes, C.P., de Sousa, A.G., da Silva, E.R., Leite, F.L., de Paiva, A.C. (2005) "On Performance Evaluation of Web GIS Applications", IEEE Sixteenth International Workshop on Database and Expert Systems Applications, 2005. Proceedings.22-26, August, pp. 497 – 501.

Chen, G. and Kotz, D. (2000) "A Survey of Context-Aware Mobile Computing Research", Technical Report TR2000-381, Dartmouth College, November.

Feng, L., Apers, P.M.G., Jonker, W. (2004) "Towards Con-text-Aware Data Management for Ambient Intelligence", In: Proc. of the 15th Intl. Conf. on Database and Expert Systems Applications, Zaragoza, Spain, pp. 422-431.

Dey, A.K., and Abowd, G.D. (2000) "CybreMinder: A Con-text-Aware System for Supporting Reminders", In the Proceedings of the 2nd International Symposium on 18 Handheld and Ubiquitous Computing (HUC2K), pp. 172-186, Bristol, UK, Springer-Verlag, September.

Dijkstra, E.W. (1959) "A Note on Two Problems in Connec-tion With Graphs", Numerische Mathematik.

Girow, A. "TinyLine: Mobile SVG software for J2ME". Available in http://www.tinyline.com. Last access in September 2005.

Gu, T., Pung, H. K., Zhang, D. Q. (2004) "Towards an OSGi-Based Infrastructure for Context-Aware Applications in Smart Homes", IEEE Pervasive Computing, Vol. 3.

Hewlett-Packard, Jena Framework. http://jena.sourceforge.net/inference. Last access 2005.

Jing. N., Huang, W., Rundensteiner, E. A. (1996) "Hierarchi-cal Optimization of Optimal Path Finding for Transporta-tion Applications", ACM Conference on Information and Knowledge Management, pp. 269-276.

Jose, R., Moreira, A., Rodrigues, H., and Davies, N. (2003) "The AROUND Architecture for Dynamic Location-Based Services", Mobile Networks and Applications 8, pp. 377-387.

Marmasse, N. and Schmandt, C. (2000) "Location-aware information delivery with ComMotion", In Proceedings of Second International Symposium on Handheld and Ubiq-uitous Computing, HUC 2000, pages 157-171, Bristol, UK, Springer Verlag, September.

Nagappan, R., Skoczylas, R., and Sriganesh, R. P. (2003) "Developing Java Web Services: Architecting and Develop-ing Secure Web Services Using Java", Indianapolis, IN: John Wiley & Sons.

OpenGeoSpatial, "Location Services". Available in http://www.opengeospatial.org. Last access in September 2005.

OpenGeoSpatial, "Web Feature Service". Available in http://www.opengeospatial.org. Last access in September 2005.

OpenGeoSpatial, "Web Map Service". Available in http://www.opengeospatial.org. Last access in September 2005.

Schiller, J. and Voisard, A. (2004) “Location-Based Services”. Morgan Kaufmann, San Francisco, pages 10-24.

Shekhar, S., Fetterer. A., Goyal, B. (1997) “Materialization Trade-Offs in Hierarchical Shortest Path Algorithms”, Symposium on Large Spatial Databases, Springer Verlag, pp. 94-111

Weißenberg, N., Voisard, A., Gartmann, R. (2004) “Using Ontologies in Personalized Mobile Applications”, In Pro-ceedings Intl. ACM GIS Conference, ACM Press, pp. 2-11.

World Wide Web Consortium (W3C), Scalable Vector Graphics (SVG) 1.1. Available in www.w3.org/TR/SVG. Last access in September 2005.

World Wide Web Consortium (W3C), Web Ontology Language (OWL). Available in http://www.w3.org/TR/owl-features/. Last access in September 2005.

World Wide Web Consortium (W3C), Web Ontology Language (OWL) Use Cases and Requirements. Available in http://www.w3.org/TR/webont-req. Last access in September 2005.

Yuan, M. J. (2003) “Enterprise J2ME: Developing Mobile Java Applications”, Prentice Hall.