Architectural Requirements for Building Context-Aware Services Platforms[1]

Patrícia Dockhorn Costa, José Gonçalves Pereira Filho[2], Marten van Sinderen
Centre for Telematics and Information Technology, University of Twente
PO Box 217, 7500 AE Enschede, The Netherlands
{dockhorn, filho, sinderen }@cs.utwente.nl

Abstract

Context-aware platforms aim at providing support to application designers to conceive their context-aware applications using services, mechanisms and interfaces that shield them from the complexity introduced by handling contextual information. This paper explores the essential requirements to be satisfied by context-aware service platforms and proposes a definition of a generic architecture supporting the execution of adaptive context-aware mobile applications. The WASP platform, a web services based context-aware service platform on top of 3G networks, is taken as a reference.

1. Introduction

Computing is moving from the traditional desktop paradigm to a mobile computing paradigm, in which new types of computing devices augment the users' workspace and the user environment changes dynamically as a consequence of the user's mobility.

This new paradigm has brought the possibility of exploring the dynamic context of the user. However, most computer systems are still designed to ignore (or assume fixed) contextual information and process their work based only on explicit input. Therefore, these systems do not take advantage of implicit input offered by the dynamic environment in order to provide added-value services or to execute more and complex tasks [3].

Context-aware computing deals with the ability of computer systems to obtain contextual knowledge in order to perform relevant tasks. Rather than treating mobility as a problem to be solved, context-aware computing seeks to exploit the nature of it. As a consequence, it creates a new generation of applications in which the user-application interaction is enhanced by perceiving/sensing the surrounding environment. It is expected that the dynamic adaptation of devices and applications in a changing physical and social environment leads to an enhancement of the user experience.

Dealing with context implies a radical design shift to cope with highly dynamic environments and changing user requirements. Because of that, in the past few years we have seen research efforts towards service platforms that provide architectural and programming support for building context-aware applications [5,6,10]. These platforms aim at providing support to application designers to conceive their context-aware applications using transparent services, mechanisms and interfaces that shield them from the complexity introduced by handling context.

In this paper we concentrate on the essential requirements to be satisfied by context-aware services platforms. The goal is to define a generic architecture for supporting the execution of adaptive context-aware mobile applications, taking as reference the WASP project [17], which aims at the development of a context-aware service platform on top of 3G networks, using Web Services technology.

The remainder of this paper is structured as follows. Section 2 briefly describes the concept of context in connection to context-aware computing and introduces the WASP project. Section 3 elaborates on the requirements. Section 4 presents the proposed architecture, Section 5 refers to related work and Section 6 concludes the paper, presenting the current project status and some final remarks.

2. Context and the WASP Project

The use of contextual information is essential to explore the possibilities of context-aware computing. Nevertheless, while it is simple to form an intuitive notion of context, elucidating a precise definition of it is challenging [4]. Although context has already been subject of investigation in different fields, particularly in artificial intelligence [13], only recently this notion has been explored for context-aware

[1] *The work described in this paper has been sponsored by Freeband Knowledge Impulse, a joint initiative of Dutch Government, knowledge institutes and industry.*
[2] *On leave from Departamento de Informática, Universidade Federal do Espírito Santo, Brazil. E-mail: zegonc@inf.ufes.br*

computing.

Most of the initial efforts for defining context in ubiquitous computing were specific for certain kinds of context - location and time being the more obvious examples. Schilit [15] claimed that the important aspects of context are *where you are*, *who you are with*, and *what resources are nearby*. More recently, Dey and Abowd [4] came up with a generic definition of context, which is *"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 used as reference in the literature of context-aware computing domain nowadays.

Handling context raises a number of challenging issues specially when applied to distributed systems (e.g., distributed context management, storage, and communication). Therefore, the enabling infrastructure plays an essential role in the accomplishment of distributed context-aware systems (e.g., CORBA, Web Services, etc.). *Web Services* [1] particularly, form a new set of technologies built upon widely supported Internet standards and it is claimed to be the dominant technology for distributed applications on the next generation Internet. However, only a few of the current proposals explore the use of Web Services as infrastructure for building distributed context-aware systems.

The WASP project [17] is concerned with the definition and validation of a service platform to facilitate the development and deployment of context-aware applications on top of 3G networks, using Web Services infrastructures. Moreover, it offers business opportunities to service providers that want to expose their services to the users of the platform. Examples of services providers are hospitals, restaurants, museums, etc.

The first target application is a tourist guide service, which will help users in typical tourist scenarios, such as visiting museums and finding a suitable restaurant based on users' profile and location. Initially, location awareness services are the research focus in the project, to test the suitability of 3G Networks as a context provider.

3. Challenges for the WASP platform

The WASP platform should provide generic functionality to address the basic challenges of context-aware computing concerning a service platform. Such functionality includes manipulation of contextual information, support for different kinds of sensing mechanisms, reactive behavior, coordination between different applications, discovery and publishing of services, support for security and privacy issues and charging. We will elaborate on these in the following sections.

3.1. Contextual Information

The platform should be capable of gathering contextual information from different sources and adapt to them according to the user needs and system capabilities. For that, common understanding of contextual information is required. Context representation and context storage are the main challenges related to contextual information.

Context representation/modeling: Not surprisingly, *contextual information modeling* has been subject of study in different areas, such as databases and artificial intelligence, for many years. In the area of artificial intelligence, there have been important findings in formalizing contextual knowledge. For instance, McCarthy's group [13] introduced logical properties of context and extended the classical logical proposition language creating the *general propositional language of context* by introducing context logical concepts. More recently, researchers in ubiquitous computing looked into formality and expressiveness to support the design of context-aware systems using conceptual modeling approaches [9].

Lately, much effort has been spent on the definition of a standardized way of representing context. For example, the W3C community developed a standard, called Composite Capabilities/ Preference Profiles (CC/PP), for describing device capabilities and preferences with a focus on wireless devices, such as PDAs and mobile phones [10]. It is based upon the Resource Description Framework (RDF), which is a technique for representing knowledge. CC/PP uses the XML serialization of RDF, one of the

many ways supported by RDF. Although CC/PP is designed to describe information about device hardware and software capabilities, it can describe a wider variety of context information, as long as that context information can be described in terms of CC/PP components and attributes (or subtypes of them).

There are, however, limitations in CC/PP which make this model not very suitable as a context model for future pervasive systems. According to [7], it becomes difficult and unintuitive to use CC/PP when the relationships and constraints in the context model are complex. A novel representation format called Comprehensive Structured Context Profiles (CSCP) has been developed by [7] and it is claimed to overcome the shortcomings of the CC/PP specification language regarding structuring.

Context storage and retrieval: Contextual information will be made available through the service platform. In order to keep track of the information in a common way, the platform will be responsible for gathering context from different sensors, processing it, and storing the results in the context storage. The storage is also important to hold contextual information over time. Therefore, information like the location of a person, at a given date in the past, can be retrieved.

In fact, different representations can be used for communication, processing and storage, each optimized for its own purpose taking into account the software and hardware that are used for this. It is important that the interpretation of these representations is unique, and that appropriate transformations are supported, i.e., the parties involved in the communication must agree upon the semantics of the information.

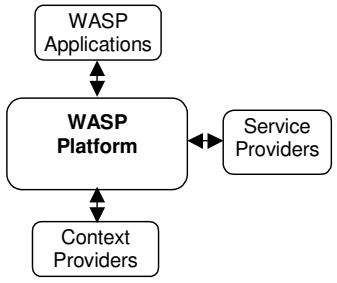

Figure 1 - Platform interactions

3.2. Platform Interactions

The platform interacts with three systems: *WASP Applications*, *Services Providers* and *Context Providers*, as depicted in Figure 1.

The next paragraphs discuss those interactions and the challenges related to them.

a) Support for different kinds of context providers (platform-context provider): The platform should be open to new kinds of third party context providers and to new kinds of sensing mechanisms, not only 3G Networks.

Context providers supply information using different communication protocols and in semantically different formats. Therefore, components that hide the process of acquiring context from sensors and providers are necessary. For this reason, the platform should have an adaptation layer that makes contextual information from different providers uniformly presented (well understood) to the rest of the platform.

An example that gives an idea of solution for this challenge can be found in the Context Toolkit conceptual framework [3], which introduces modeling abstraction elements such as Widgets. Widgets encapsulate sensors providing semantically uniform operations to access context. For instance, a Widget could be responsible for translating latitude and longitude to a street name and a house number.

b) Reactive behavior (platform-application): Applications should be able to respond to their dynamic environment. Therefore, the platform has to support the applications in this process since the platform is the one aware of the changes in the user environment. But how to support a large and growing number of different context-aware applications without having to upgrade the platform each time a new application is deployed? One possible solution is to give some intelligence to the platform by exposing reaction mechanisms, i.e., applications have to "teach" the platform how to react to certain correlations of events [8]. Suppose E_1, E_2, ..., E_n are known events to the platform. A possible correlation of these events is $((E_1 \text{ OR } E_2) \text{ AND } E_5) \text{ OR NOT } E_6$. On one hand, if the result of the formula turns to be true, it enables some kind of action to be triggered. On

the other hand, if the result turns to be false, it disables the action to be triggered.

Figure 2 depicts the desired architectural models of interaction to be supported by the system, abstracting from the 3G networks. The upper layer represents the application layer and the lower, the supporting platform. The request/response model (passive platform) is the one where the reactive behavior of the platform is just to respond to the application requests. In the event-driven model (event-driven platform), applications expose to the platform the desired reactive behavior by means of a subscription. The subscription is based on correlation of events and programming of actions. In this paper we consider the event-driven model, i.e., the platform is programmed to react to a certain correlation of events.

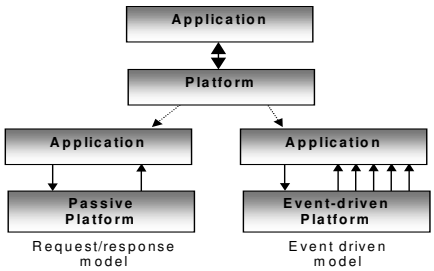

Figure 2 - Different models of interaction between application and platform

For instance, consider the simple application "*Send me a reminder of buying bread when passing close by a bakery*". The platform understands user and bakery as *entities*. *Close by* can be understood as less than 200 meters of distance, depending on what the application defines. Finally, *Send a reminder* is an application action, i.e., how the application has to react on behalf of its users.

Something different is how the platform has to react to the application. This reaction from the platform is programmed by the application. In the bakery scenario, possible reactions are *send a message* or *send a list of closest bakeries* or others.

The platform has to keep track of all possible entities (user, bakery, museum, restaurant, etc.) involved in some event correlation as well as their context. In the aforementioned case, based on the user location, the platform knows how to find the bakeries in its environment.

The platform supports many applications and each of them has several subscriptions. Each subscription contains at least one trigger and when the conditional expression is satisfied, the action is triggered. Description languages such as XML are shown to be suitable to represent this kind of event correlation. The Rule Markup Language (RuleML) [2], for instance, tries to represent logical expressions, more specifically Prolog expressions, in the form of XML data and XML Schemas. Investigations are being carried out in order to test the suitability of this language for representing event-condition-action rules in the WASP project.

c) Coordination among different applications (platform-application): Different reaction mechanisms programmed by different applications can give rise to conflicting problems when providing a service for the same user. For instance, consider a device which has two different applications, a reminder and a sleep mode application. The sleep mode application turns the device to a sleep mode under certain condition, for example when the user (and the device) is inside the movies, in a meeting or when he/she is driving. The reminder sends reminder messages when a correlation of events, determined by the user, happens. The sequence diagram in Figure 3 illustrates this situation.

A component with cross-knowledge capabilities, i.e., a component with knowledge of different sources in the architecture, for example a monitoring component, can be responsible for managing the coordination of events by checking the user context and preferences. This way, side effects generated by conflicting reaction mechanisms can be avoided.

d) Discovery and publishing of services (platform-services provider): Service provision by third party service providers is the essence of the service platform. Discovery and publishing of services can be done by internal elements or/and by external elements, being opened and shared with others application environments/platforms.

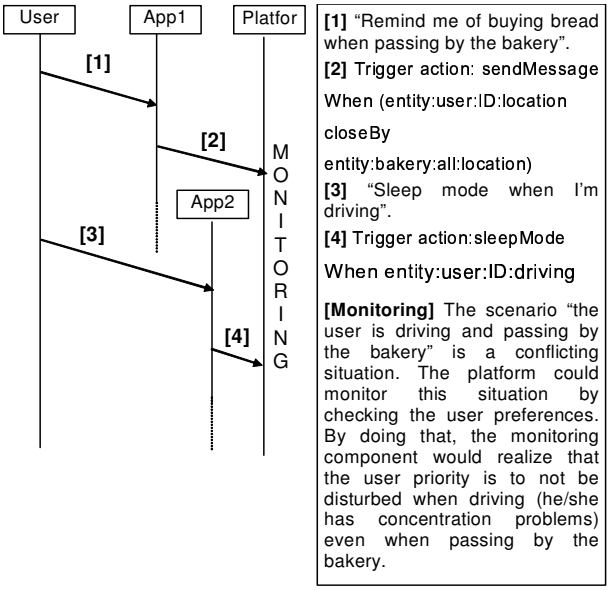

Figure 3 - Bakery scenario

Entities like museums, restaurants, supermarkets, bakeries, schools and hospitals want to expose their services to the platform, so that, depending on the user needs and preferences, and taking into account his/her context, those services can be used. In the aforementioned example "remind me of buying bread", the platform uses information about bakeries, in this case, their location. There are cases in which the platform is programmed to directly use the services of service providers. For example, the platform could be programmed to immediately order the bread when the user is approaching the bakery. Therefore, the user does not need to be bothered with selecting and paying activities; this could be automatically done by the platform.

A centralized service discovery approach, which has been largely explored in recent works, is Universal Discovery, Description and Integration (UDDI) [16]. It provides a directory service where service providers and service requestors come together to satisfy their needs. Related works inside the WASP project intend to add functionality to the UDDI in order to improve its capabilities [14]. They claim that UDDI, among other things, lacks semantic description, process specification and ontology support. Their aim is to implement an enhanced UDDI server, capable of storing, matching and retrieving semantically rich service profiles that contain contextual information.

3.3. Privacy concerns

The platform should be able to gather important and perhaps private information from different parties. Therefore, security and privacy services are clearly a necessity in a context-aware environment. Although privacy is a very important issue, it has not been properly enforced in available context-aware platforms [5,6,10]. Examples of recent work on this topic can be found in [12].

There are efforts in the WASP project to define a privacy architecture based on an extension of the Privacy Preferences Project protocol (P3P). This architecture is part of a compulsory security framework that allows the use of appropriate security policies and authorization services.

3.4. Other Challenges

There are other important architectural challenges related to charging, scalability and use of standards.

For instance, the platform will potentially manage a large volume of context information, user profiles and preferences, and it also has to keep track of a large number of event-correlated rules, which means intelligent manipulation of data from different sources in the platform. Charging is a critical service and should be based on a general business model that defines assignment of business responsibilities between the different parties involved in the platform.

A further discussion of these challenges is beyond the scope of this paper.

4. The WASP Platform Architecture

This section presents an architecture for the WASP platform based on the main issues discussed so far. Figure 4 depicts the high level view of the proposed architecture, which is composed of three main modules: *Monitor*, *Repositories* and *Context Interpreter*. We will elaborate on them in the next subsections. Security & Privacy and Charging modules are not shown in the figure.

4.1. Context Interpreter

The Context Interpreter gathers contextual information from different context providers (sensors or third parties context providers), which may use different communication protocols and

different contextual representations, making contextual information uniformly available to the platform. The context interpreter gathers and provides contextual information at different semantic levels, and it may infer new contexts from aggregating or interpreting lower level contexts. For example, the interpreter is able to assess the speed of a user from the latitude and longitude changes over time. Moreover, the context interpreter may be able to infer from this that the user is driving a car.

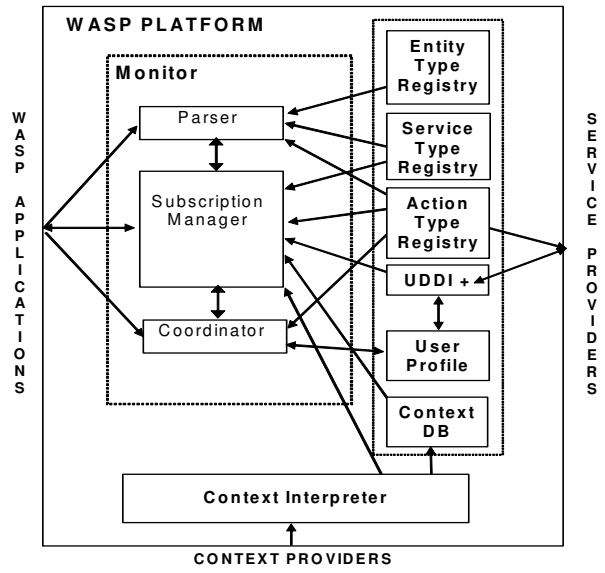

Figure 4 - WASP Platform architecture

Figure 5 depicts the internal model of the context interpreter, in which several levels for gathering and providing context are shown. Any piece of contextual information provided by the interpreter is called primitive context.

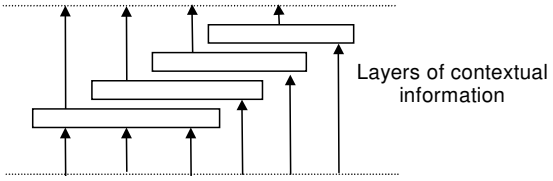

Figure 5 - Internal view of the interpreter component

In this model, the upper layers can access all the layers below it, not just the one immediately below. In fact, the context interpreter takes raw context from the context provider, semantically interprets it and then, based on this interpretation, infers new contexts. This process can be performed as many times as necessary.

The context interpreter uses three different models of information provisioning: *request-response*, *time-driven* and *event-driven* model. In the first model, the interpreter provides context only on explicitly request. In the second, the interpreter is programmed to provide the context at specific time intervals. Finally, in the last one, the interpreter is programmed to provide the context only when the context is changed.

4.2. Repositories

These architectural components support the Monitor in the management of subscriptions.

Entity Type Registry: Repository of entity types (and their attributes) registered in the platform. Examples of entity types are user, bakery, hospital, restaurant, etc. Examples of attributes are *MobileNature* (mobile or fixed) and *Context* (location, velocity etc.). It is possible to apply different kinds of context for different entity types. *Velocity*, for instance, is a context applied to users but not to hospitals or bakeries. The entity type registry needs to keep track of all possible combinations of context and entity types.

Service Type Registry: Repository for storing information about the types of services supported by the platform. A service is applicable for certain number and types of parameters. The service *CloseBy* (proximity of two or more entities) makes only sense when two or more entities are involved. It also only makes sense if the contextual information location is applicable for those entities.

Action Type Registry: Repository for storing information about types of actions. Actions are tasks performed in response to an enabling application's subscription. Like services, actions differentiate in number and types of parameters. An example is the action *SendMessage* mentioned in the bakery example. This action should have as parameters the recipients of the message and the contents of the message, which could be the location of the closest bakeries. There are situations in which actions are mutually exclusive, i.e., they cannot be performed at the same time. For instance, the actions *SendMessage* and *MessageOFF* applied to the same user can generate a conflicting situation. For this reason, the platform needs to know which actions are conflicting and this information is made available in the action type registry.

UDDI +: Component that deals with the *discovery and publishing of services* requirement. It is responsible for storing, matching and retrieving semantically rich service profiles. In the bakery example, the UDDI+ stores the location of the bakeries, their specialty and their service time (opening and closing times). Moreover, given the location of the user, the UDDI+ is capable of returning all the bakeries that are close by and it is also capable of performing intelligent matching using the user preferences profile (e.g., selecting the bakery whose specialty best suits the user's taste or diet).

User Profile: Component responsible for storing and managing user profiles. Therefore, it is consulted by the UDDI+ when matching user preferences with respect to services. It is also used to make decisions when a conflicting situation occurs between different applications.

ContextDB: Component that handles the *context storage/retrieval* requirement. It is responsible for keeping track of contextual information of the entities. It gathers contextual information from the context interpreter.

4.3. Monitor

The core of the platform architecture is the Monitor module, which tackles the requirements *reactive behavior* and *coordination among different applications*. This module is responsible for interpreting and managing the applications' subscriptions. In order to perform its operations, the monitor makes use of the data available in the repositories and the contextual information provided by the interpreter.

A subscription provides the means to dynamically configure interactions between applications and platform. An application's subscription is an enabling expression, i.e., the result of the expression must be true or false. The possible elements in a subscription are primitive contexts, logical operators, services and primitive values. Primitive contexts are any known piece of contextual information, logical operators are AND, OR and NOT, services are special tasks performed by the platform and primitive values can be numbers, letters, constants, current time etc. We will elaborate on this when explaining the internal components of the monitor. The subcomponents of the monitor are *Parser*, *Subscription Manager* and *Coordinator*:

Parser: This component is responsible for verifying if the subscription is syntactically and semantically correct. To perform this task, it makes use of the repositories *Entity Type*, *Service Type* and *Action Type*. The result of the parsing is a tree of primitive contexts, services and logical operators. The example tree depicted in Figure 6 is the parsing result of the following subscription:

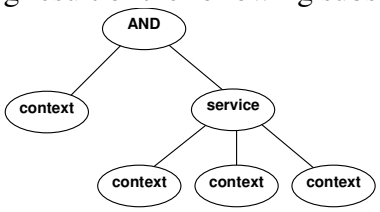

Figure 6 - Example of a parsed subscription

```
Trigger action:sendMessage (user:1,
                        user:2, user:3)
When (entity:user:1:driving AND
      closeBy
          (entity:user:1:location,
           entity:user:2:location,
           entity:user:3:location)
```

Subscription Manager: Once the subscription is parsed and the tree is built, the subscription manager keeps track of the enabling and disabling conditions in order to trigger (or not) the action. This means constant check of the involved contexts and performing of the services. In order to perform its operations, the subscription manager gathers contextual information from the context interpreter and from the *contextDB*. When service providers are involved in the subscription by means of their context or services, the UDDI+ component is also called by the subscription manager. The registries *ActionType* and *ServiceType* are also checked to correctly perform the services and to trigger the actions.

Coordinator: This component is responsible for handling conflicting subscriptions. A subscription is conflicting when the involved actions are mutually exclusive. The coordinator consults the action type registry to verify which subscriptions are conflicting. By doing this and also by checking the user priorities, the coordinator is able to choose one of the actions.

5. Related Work

There have been several efforts aiming at the development of context-aware platforms. Indulska et al. [11] present an architecture that focuses on the handling of different types of adaptation mechanisms for device-oriented services and resource availability/adaptability such as bandwidth and power energy. Efstratiou et al. [6] developed a specific language for coordination of events intended for enterprise domain. DeVaul et al. [5] describe an infrastructure based on a distributed database and a dynamic decentralized resource discovery service in the area of wearable computing.

Differently from the aforementioned research efforts, the WASP project focuses on facilitating the development/deployment of context-aware applications using a subscription language which allows dynamically configuration of interactions between applications and platform, on top of 3G networks. Moreover, it explores the web services' service discovery approach (UDDI+) [14].

6. Conclusions

This paper outlines some of the technical challenges related to the design of context-aware services platforms, and proposes a generic platform architecture for supporting the development of context-aware applications.

Some components of the WASP platform have been prototyped (UDDI+, Location Interpreter), initially focusing on the development of location-aware applications. The subscription language, which enables application-platform interactions dynamically configurable during the platform run-time, is currently being defined. The further development of the platform will allow generic deployment of a larger range of context-aware applications addressing different types of contextual information and providing support for both application-platform and platform-context provider interactions using the event-driven interaction model.

Bibliography

[1] Almeida et al., Web Services Technologies. *WASP Deliverable: D3.1*, January 2003.

[2] Boley, H., The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations. *14th Intl. Conf. of Applications of Prolog (INAP2001)*, Univ. of Tokyo, October 2001.

[3] Dey, K., Abowd, D., A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction Journal* 16, 24 (2001), pp. 97-166.

[4] Dey, A. et al., Towards a Better Understanding of Context and Context-Awareness. *Technical Report* 99-22, Georgia Institute of Technology, 1999.

[5] DeVaul, R. et al., The Ektara Architecture. *MIT Technical Report*, 2000.

[6] Efstratiou, C. et al., An Architecture for the Effective Support of Adaptive Context-Aware Applications. *Mobile Data Management*, 2001.

[7] Held, A. et al., Modeling of context information for pervasive computing applications. *Proc. of the 6th World Multiconf. on Systemics, Cybernetics and Informatics (SCI2002)*, Orlando, FL, July 2002.

[8] Henderson, M., *A Framework for Event Correlation.* Master Thesis, Department of Computer Science and Electrical Engineering, University of Queensland, October 1999.

[9] Henricksen K., et al., Generating Context Management Infrastructure from High-Level Context Models. *Proc. of the 4th Intl. Conf. on Mobile Data Management, Industrial Track Proceedings*. January 2003, Melbourne, Australia, pp. 1-6.

[10] Indulska J. et. al., Experiences in Using CC/PP in Context-Aware Systems. *Proc. of the 4th Intl. Conf. on Mobile Data Management*, January, 2003, Melbourne, Australia, pp. 247-261.

[11] Indulska, J. et al., An Open Architecture for Pervasive System. *Proc. of the 3rd Int. Working Conf. on Distributed Applications and Interoperable Systems (DAIS 2001)*, Kraków, Poland, pp. 175-188.

[12] Langheinrich, M., A privacy Awareness System for Ubiquitous Computing Environment. *UbiComp* 2002, Springer LNCS 2498, pp. 237-245.

[13] McCarthy, J., Notes on formalizing context. *Proc. of the 13th Intl. Joint Conf. on Artificial Intelligence (IJCAI 1993)*, Mountain View, CA, USA, 1993.

[14] Pokraev, S. et al., Extending UDDI with context-aware features based on semantic service description. To appear in *1st Intl. Conf. on Web Services (ICWS 2003)*, Las Vegas, USA, June 2003.

[15]Schilit, B., et al., Disseminating Active Map Information to Mobile Hosts. *IEEE Networks*, 8(5) (1994), pp. 22-32.

[16] UDDI project. *UDDI: Specifications*. [http://www. uddi.org/specification.html].

[17]WASP project [http://www.freeband.nl/projecten /wasp/ENindex.html].