Design of Context-Aware Applications Based on Web Services

T. Chaari, F. Laforest
LIRIS
INSA Lyon, France
{tarak.chaari,frederique.laforest}@insa-lyon.fr

A. Celentano*
Dipartimento di Informatica
Università Ca' Foscari di Venezia, Italia
auce@dsi.unive.it

Università Ca'Foscari di Venezia reference: Technical Report CS-2004-5

ABSTRACT

Context-aware systems are applications that adapt to several situations involving user, network, data, hardware and the application itself. Researchers in context-awareness have concentrated on how to capture context data and to carry it to the application. In this paper, we study the impact of context on the core of the application. First, we give the guidelines of a context model based on a more practical definition of the context. Then, we propose a context-aware architecture providing a functional adaptation to the context.

1 INTRODUCTION

Data centered applications exchange information with users at different levels of detail, content and presentation according to several parameters which depend on the user and his/her environment. They can also provide, within the same application, different services to different users, or to the same user in different situations. Hence, the notion of *context* has been developed, as a means to adapt the behavior and the interface of an application to the user situation and equipment, encompassing a large range of adaptation parameters.

In different contexts, users may access different data and exploit different aspects of an application. For example, in one context a doctor accesses a health database for screening patients for prevention cares, while in a different context the same doctor accesses the same database for post-treatment analysis of cases. While the data are the same, the way they are returned may vary according to the doctor's goal. Often, in different contexts, users access almost the same data and the same services but receive answers shaped differently, with different presentation and possibly different content detail. For example, a doctor examines a patient record at the hospital using a desktop computer connected to the hospital database, or consults the same record stored on a PDA while visiting the patient at home, or receives an audio description of the patient record during a surgical operation.

Context-awareness and adaptation are tightly related, and the two terms are often used as synonyms. However, they refer to different capabilities: *adaptation* is the capability to provide different versions of a service or different presentations of a document, in order to suite the needs of the user, of the environment, of the equipment, etc.; *context-awareness* is the capability of perceiving the user situation in its many aspects, and of adapting as a consequence the system behavior, i.e., the services, the data and the interface. Adaptation is therefore the goal of context-awareness, which is able to drive it without explicit tuning operations by the user.

A context-aware application must manage the context as one of its inputs, processing any user request according to the different context instances. However, it is simplistic to consider the context as one of the application input data, due to the difficulty of classifying in advance all the combinations of user situation, equipment and other context-dependent parameters. Context, therefore, must be managed separately and its influence on the application behavior must be described orthogonally with respect to the application data.

The paper is organized as follows: after reviewing the state of art in context-awareness in Section 2, in Section 3 we briefly discuss context modeling. A context-aware architecture based on Web services is proposed in Section 4. Section 5 discusses the structure and the composition of adaptation services, with a focus on adaptation of the application's behavior. Section 6 draws the conclusions and the future work.

2 STATE OF ART

Research on context-awareness has started with addressing the problem of mobility by hiding it to the user. The first real context-aware computing effort was initiated by Researches at Olivetti Research Ltd and Xerox PARC Laboratory [19]. Since then, many other researches have studied this topic and contributed to this domain. Early works approached the problem by studying location-awareness, and still now many context-aware applications are limited in the scope of their analysis, using small pieces of contextual information and presenting ad hoc solutions for very specific needs.

In most contributions in this area, we distinguish three main steps that an application has to do in order to be context-aware. First, we have to capture low level contextual information from different sensors (for example, GPS coordinates). Second, we have to make some interpretation on what we capture to build high level contextual information which is more relevant to the application. For example, we can transform GPS coordinates to a complete address and compute physical, temporal and semantic relationships from the initial low level context values. Finally, we have to carry this interpreted information to the application. The Context Toolkit [7] is one of the first context-aware architectures considering these three main steps. Sensors capture low level context signals and present them to the context widget; widgets use the interpreter to carry high level context data to the application through a context server.

Some researches store the context before its dissemination to build a contextual history. This step is very important to compute the high level representations of the context. For example, we may require the last location of the user before arriving to the current one, in order to analyze the user motion.

This extra step has revealed another potential need in context

* Work done during a sabbatical leave at INSA Lyon, France

research, which is context modeling. In fact, before storing the context, we have to find a reliable representation of all its aspects.

We distinguish three approaches in context modeling.

- The first approach stores context as a simple set of attribute/value pairs, e.g., {Name="context1", User="x", Location="y", Time="t"). The Context Toolkit uses this approach.
- The second approach presents the context using RDF. The most consistent is an extension of the CC/PP W3C profile [15] called Comprehensive Structured Context Profiles (CSCP), proposed by Held in [13].
- The third approach models the context using ontologies [4]. The most consistent one is CoOL [18] that presents a context parameter as a set of entities having certain aspects representing its characteristics.

Table 1 presents a summary of these approaches with the advantages and the drawbacks of each one.

Table 1. Existing context models

Model characteristics	Expressiveness and semantic richness	Implementation ease	Conflict resistance
Attribute/value pairs	-	+	-
RDF based	+	+	-
Ontologies	+	-	+

After modeling and storing the context, it must be carried to the application, with information about how the application can adapt to context changes. In this area, Dockhorn Costa [8] distinguishes between four research approaches:

- *Conceptual frameworks* focus on the architectural aspect of context-aware systems and provide means to facilitate capturing, interpreting and carrying context data to the interested parties. The Context Toolkit [7] and the Cooltown [16] projects are examples of this approach.
- *Service platforms* aim at providing the pertinent services to the user depending on context. This includes dynamic service discovery, dynamic deployment of adaptive services addressing issues of scalability, security and privacy. M3 [14] and Platform for Adaptive Applications [9] are examples of contributions to this approach.
- *Appliance environments* try giving solutions to the heterogeneity problem by providing interoperability techniques and frameworks. Ektara [5] and Universal Information Appliance [11] are projects which use this approach.
- *Computing environments* for pervasive applications focus on designing the physical and logical infrastructure to hold ubiquitous systems. The PIMA [2] and Portolano [10] projects are examples of this approach.

Table 2 presents a synthetic view of these approaches by comparing the most relevant issues.

In conclusion, we can say that a practical complete context model is yet missing. Moreover, in the existing context-aware applications, there is a great interest to how to gather the context and how to carry it to the system, but there is no consistent answer to the question: How the application can adapt to the context? To be more precise, we reformulate this question: What is the impact of context on the application core?

Table 2. Approaches in context-awareness

Issue	Conceptual Frameworks	Service Platforms	Appliance Environments	Computing Environments
Device heterogeneity			X	
Device mobility			X	
Context management	X	X		
Adaptation		X		X
RAD/ deployment		X		X
User context	X			

3 CONTEXT MODELING

Context-aware applications usually mix context management code within the application code. The application code becomes more complex and more difficult to read and maintain. Decoupling context-independent activities of the application from contextual concerns would locally reduce the code complexity. In this article, we present solutions to make this effective. The first step, discussed in this section, concerns the separation of contextual data from application data, while the second step concerns the application architecture, and will be discussed in the next section.

Practical definitions of context — and more precisely of context data — have not yet drawn to a consensus. Definitions in the literature are often domain-oriented and thus too limited. Dey defines the context as "any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object" [6]. It is a complete and general definition, but it does not help in separating the contextual data from the application data. We believe that this distinction is important because it reduces the complexity of the design of the final application. The core of the application should be designed by making abstraction of any context, which should be included in a second step. Such a way of working allows turning a legacy application into a context-aware one, leaving the legacy application unmodified.

To do so, the boundary between contextual data and application data has to be clearly defined. It depends on the application domain, since some data that of the application level in one domain can be seen as context in another domain. For example, GPS localization is context data in a telemedicine application, but is application data in a traffic regulation system.

We can define the context as the set of external parameters that can influence the behavior of the application. Context parameters evolve during application runtime; they are not significant to the user, therefore they must be transparent to him/her. A new instance of these parameters characterizes a new context situation, which does not modify the application data, but may select of process it in a different way. For example, in a home care application, when a doctor moves from a patient residence to another, the patient records do not change; they are part of

application data. However, the current patient *id* changes; it is therefore a context parameter.

4 A CONTEXT-AWARE ARCHITECTURE BASED ON WEB SERVICES

The development of context-aware, adaptable applications requires two goals to be assessed: (1) how to design an architecture supporting context-awareness at run-time, and (2) how to design the application itself in order to be context-aware. Different technologies can be used to build applications, and no one appears to dominate the current scenarios. The adoption of Web services, however, is widespread and is considered today a viable architecture for evolving applications, mainly due to its "loosely coupling" approach to integration of application functions [1]. We therefore adopt a Web services paradigm for discussing about context-aware application design.

Figure 1 illustrates a service oriented architecture, in which the components devoted to context management are separated from the application core. Each component corresponds to a phase of context management, as discussed in the following.

Context capturing. Context capturing concerns physical sensors and the raw data they generate. This part is highly device-dependent and a generic model is difficult to build. In our architecture we define a *context provider* that represents a context capturing system.

Context interpreting. The low level representations that we initially capture may not be meaningful to the application, while high level representations are easier to interpret and to use (e.g., an address is more significant than GPS coordinates). In our architecture, the *context interpreter* module makes the context interpretation.

Context modeling. A *context repository* stores context values. To model context parameters, we use XML documents storing and exchanging context values. We define a set of elements for each context facet (user, network, device, metadata), containing current values of parameters relevant to that facet. Defining the set of context parameters is the job of the application designer, who defines also their syntactic and semantic structure (e.g., scalar values, sets, references to other parameters,…).

Context dissemination. A context aware application has to consume a part of the context. In a service oriented architecture it must subscribe to the context broker that carries the pertinent data to each service in the application. While subscribing, the service tells the broker which part of the context is relevant to it. Then, the broker can provide a context view for each service. This view can dynamically evolve during execution, requiring some intelligence in the broker. Services may pull context values each time they require it, or the context broker may push context to subscribers every time it is updated.

Adaptation to the context. Context consumers have to adapt to the context. To do so, they are first registered to the context broker. Context adaptation can concern three levels: data flow (content adaptation), visualization (user interface adaptation), and application's behavior (service adaptation). All these adaptations can be static (before runtime) or dynamic (at runtime). We must use both to ensure the best adaptation to the context. Content and user interface adaptations are well studied in the literature, even if there are still aspects that need to be explored more deeply. In the next section we shall discuss mainly the application's behavior adaptation, and only survey content and user interface issues.

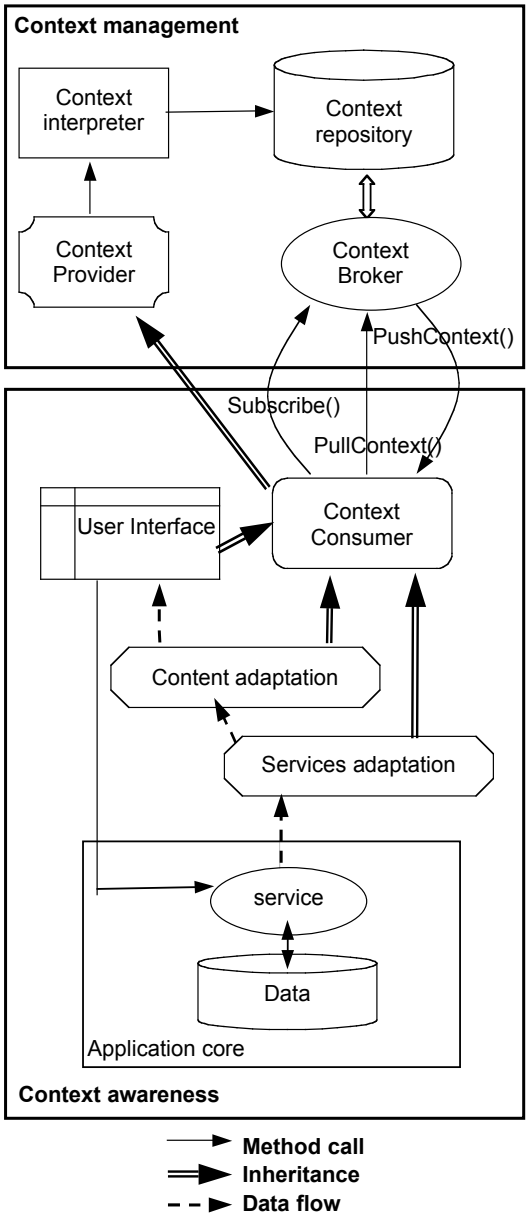

Figure 1. A service oriented architecture

5 ADAPTATION SERVICES

It would be desirable to achieve a certain degree of independence in composing services, at least at the outermost levels; i.e., adaptation processing should be done by different combinations of services which are independently selected on the different dimensions of the application (user interface, content and business services), as shown in Figure 2.

Web services can be used to ensure the adaptation of the three dimensions of the application. Since we assume that the business core of the application is implemented by Web services, such a way of working guarantees a variable degree of granularity at one hand, and a flexible and reusable application assembly and deployment at another hand. It also helps carrying out the context adaptation at many levels of the business core.

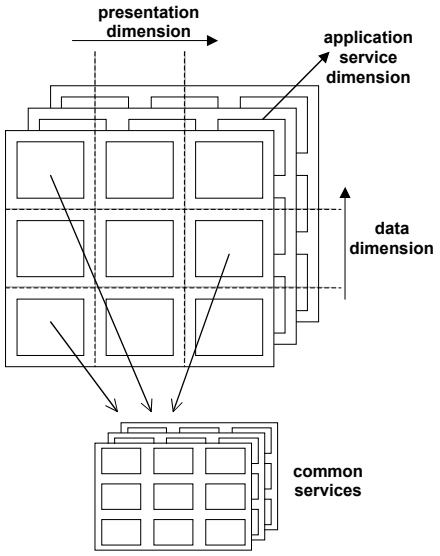

Figure 2. Three dimensions for application adaptation to the context

In Figure 2, a point in the 3D space defines an adaptation plan, i.e., a data adaptation route, plus an application service adaptation route, plus a presentation adaptation route. The three routes are planned in common, but may be processed independently. Results are gathered to build the context adaptation. Each service can access a pool of common services which can also be context dependent. Common services are "general" adaptation services (e.g., image compression).

To be efficient and ready for adding new contexts with limited effort, each dimension combination should not be completely pre-built, but should result from the dynamic combination of independent services, each related to its own dimension.

5.1 User interface

It is the most important part of the application to the user. User interfaces present data to the user in different modalities according to the context situation. We use the MVC model [17] to design the user interfaces. An interface consists of a view, a model and a controller. The model is the set of business services that interact with the view and the controller links between the services and the interaction components of the view. It detects the interaction events from the view and invokes the corresponding services in the model. The adaptation of user interfaces mainly depends on the device capability and the user facets of the context. We have proposed a framework — based on adaptation rules at runtime and before runtime — to adapt user interfaces to these two facets [3]. This framework allows the user to describe the desired user interface for any web service. The user begins by choosing the abstract widgets that he/she wants to have on the display; then he/she selects a service from the services registry of the application. Finally the whole code of the user interface is generated to the specific terminal of the user. While calling a service, its inputs and outputs are also adapted to the context situation.

5.2 Content adaptation

Content adaptation consists in modifying some properties of the data delivered to the user. In this section we give a non exhaustive list of possible transformations to ensure content adaptations; as we said in the introduction of this section, every transformation can be implemented with Web services.

Format adaptation. This transformation consists in modifying or completely changing the data format. For example, reducing the number of colors of an image or synthesizing voice from a text.

Language translation. A text can be translated to another language to adapt to the user preferences. Translation services can provide this type of adaptation.

Data compression. We distinguish two main types of data compression: raw data compression (e.g., zip) for reducing storage size and transmission time, and multimedia compression (e.g., jpeg) which is directly processed on the user terminal. For text data, semantic compression can be used to compute a summary of a document.

Data decomposition and aggregation. Data decomposition consists in extracting some part of a media that is more preferable or more significant to the user. The remaining part can be displayed to the user if the terminal supports it. The user can demand an aggregation of different media objects: e.g., a tourist can demand a part of a map (decomposition) with a list of the restaurants nearby (an aggregation of two media objects). The decomposition and the aggregation transformations can also be temporal; e.g., the user can be interested only in a few animated sequences from a large video file. The service can present them separately or together, after their aggregation.

5.3 Application's behavior adaptation

From the perspective of the functional architecture, an application is based on a number of business services that perform the desired functionalities of the application. Business services are not context-aware and their behavior remains the same in the different context situations. In such applications each service is described by a function $f(x)$ getting input x and computing some output values. Services may access the data storage to produce changes in the application state.

A first adaptation level, the *internal adaptation*, can be done by introducing context data as an extra entry to services: each service manages context data and application data. Context data ensure the functional adaptation in the built-in code of the service. Application data are used in each code bloc for real business service job. This can be represented as in Figure 3.

Internal adaptation uses functions that can be defined as $f(x, cf(c))$, where x represents application data, and $cf(c)$ selects the part of the context situation that concerns the function f. The cf function is managed by the context broker that selects the useful context data for function f.

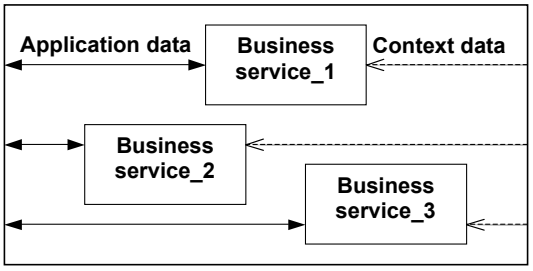

Figure 3. Internal adaptation of services

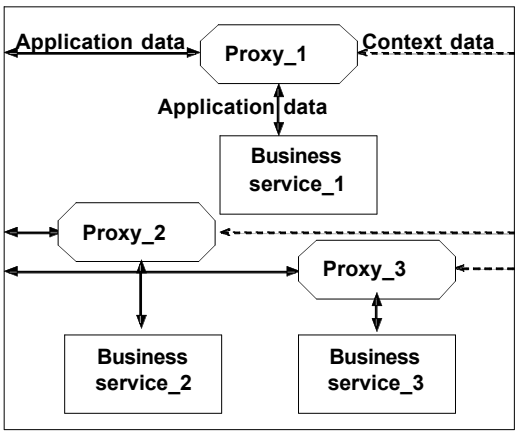

Figure 4. External adaptation of services

A second adaptation level, the *external adaptation*, externalizes context data management from application data management, by providing an intermediate service (*proxy service*) on the business services (Figure 4). The proxy gets application data and context data. It makes modifications on inputs and outputs of the business service according to the context situation. In this case the business service is not context-aware: it gets application data only.

External adaptation uses functions that can be written $p(x, cf(c))/f$, where p is the proxy function that gets the application's data x and $cf(c)$ is the part of context concerning the service f. p ensures the adaptation of the inputs and the outputs of the business service f. The notation "p/f" means "*p knowing f*". f is not a real entry parameter of p but it can be considered as one of its members.

A third adaptation level, the *polymorphic adaptation*, concerns the selection of service instances or versions according to the context situation (Figure 5). Each service has different versions or possible instances for different context situations (a, b, ...), but some services may not exist in all versions. The selection of a service among the available versions is made by a *tier* service, as it is done in the *design pattern* strategy [12]. In general, different adaptations are not completely equivalent at the outer level, e.g., due to the different types of handled data. It can be written as $s(x, cf(c))/f_a, f_b, ...$, where s is the *strategist service* that chooses among the f_i according to the current context situation, x is the application data, s knows the list $f_a, f_b, ...$, of the available strategies for a given service f.

The situation is really more general, since services can use other services, in a nested style. The execution of a service therefore may be the result of a composition of services in the style $f(g[h(..., ch(c)), cg(c)], cf(c))$. Each level of composition requires its context values, using its context selection from the broker.

The composition is not the only adaptation operator on services. In the remainder of this section we present a non exhaustive list of adaptation operators using the following notations: $F = \{f_1, f_2, \ldots, f_n\}$ is the functional core of the application composed by the set of its business services. Every service f_i can be polymorphic and can have many possible instances or versions $\{f_{ia}, f_{ib}, ...\}$. $R = \{r_1, r_2, \ldots, r_n\}$ is the set of data composing the output of a service f. Each elementary output r_i can have many instances $\{r_{ia}, r_{ib}, \ldots\}$.

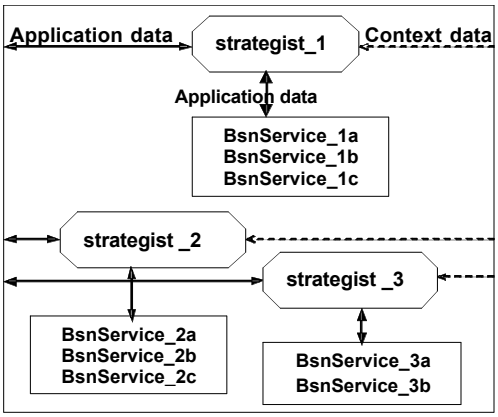

Figure 5. Polymorphic adaptation of services

Projection (Π)

This operator can be applied to the list of the business services of the application. It consists in removing the access to some services, e.g., for compatibility with user preferences or access rights. E.g., let $F = \{f_1, f_2, \ldots, f_n\}$ be the set of business services of the application, and $F' = \Pi(\{f_2, \ldots, f_n\}, F)$. The resulting functional core F' doesn't include the f_1 service. F' is a projection of the original functional core on $\{f_2, \ldots, f_n\}$.

This operator can also be applied to the output data of the service, e.g., $R' = \Pi(\{r_2, \ldots, r_n\}, R) = \{r_2, \ldots, r_n\}$. In this case, the elementary output information r_1 of the corresponding service is not returned to the user. This can be useful if we want to hide some output values, or to prepare an entry data set for another service that takes only a part of the output of the previous service.

Join (join)

This operator can be used to augment the functional core of the application F by an additional service h. $F' = join(F, h) = \{f_1, f_2, \ldots, f_n, h\}$. This operator can be applied to a set of input or output data. In this case it is equivalent to the *join* operator of a relational database.

Composition (o)

The composition can be applied to two services f_1 and f_2 and can be serial (o_s), if the services are applied sequentially, or parallel (o_p), if the services are applied independently and the results are joined. Note that in general the serial composition is not commutative: $f_1\ o_s f_2$ is different from $f_2\ o_s f_1$.

Selection or restriction (σ)

This operator can be used to remove some possible instantiations $\{f_{ia}, f_{ib}, ...\}$ of a certain service f_i. For example, $\sigma(\{f_{ia}, f_{ib}\}, f_i) = \{f_{ia}, f_{ib}\}$. In this case, f_{ia} and f_{ib} are he only possible instances of f_i. The selection can be applied to the outputs R of a service to remove some non relevant instances of the output data.

Union (∪)

This operator can be used to add another possible instance or version of a polymorph service: $\cup(f_i, f_{i\alpha}) = \{f_{i\alpha}, f_{ia}, f_{ib}, \ldots\}$. $\{f_{ia}, f_{ib}, \ldots\}$ are the initial possible instances or versions of f_i. This operator can be applied to a set of input or ouput data of a service. In this case, it is equivalent to the union operation in the relational database domain.

6 CONCLUSIONS

In this paper, we have proposed a new definition of the context that separates application data from context parameters. We have classified context parameters in four facets (user, network, device capability and metadata) to simplify its management and its storage. The new definition helped us to make a more precise study on how to guarantee context-awareness in the application core. Finally we have given the guidelines of adapting the three dimensions of an application (data, presentation and behavior) to the context. We have proposed a solution based on Web services to guarantee these adaptations. A set of operators on these services ensures the context-awareness inside the application core.

The proposal is complete from the conceptual and methodological point of views, but needs to be evaluated in a set of experiments. Therefore, we shall define some concrete rules to map the context situations to an adaptation route using a combination of our operators. Then, we shall develop a prototype following the schema of Figure 1 and including the behavioral adaptation detailed in Section 5. Among the domains where such adaptation can be successfully tested, we plan to approach the telemedicine domain, where the variance in offer of services and the needs of its different users make adaptation a valuable goal.

7 REFERENCES

[1] D. Austin, A. Barbir, C. Ferris, S. Garg. Web Services Architecture Requirements, W3C Working Draft, 19 August 2002, http://www.w3.org/TR/2002/WD-wsa-regs20020819.

[2] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman and D. Zukowski. Challenges: An Application Model for Pervasive Computing. *Proc. 6th Annual Intl. Conf. on Mobile Computing and Networking (MobiCom 2000)*, pp. 266-274, Massachusetts, USA, August 2000.

[3] ***. Génération et adaptation automatiques des interfaces utilisateurs pour des environnements multi-terminaux. Le projet SEFAGI : Simple Environment For Adaptable Graphical Interfaces. ***

[4] H. Chen, T. Finin and A. Joshi. An Ontology for Context-Aware Pervasive Computing Environments. *IJCAI Workshop on Ontologies and Distributed Systems, IJCAI 2003*, Acapulco, Mexico, 2003.

[5] R. W. DeVaul, A. S. Pentland. *The Ektara Architecture: The Right Framework for Context-Aware Wearable and Ubiquitous Computing Applications*. MIT Technical Report, 2000.

[6] A. K. Dey, G. D. Abowd. Towards a Better Understanding of Context and Context-Awareness. *CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness*, The Hague, The Netherlands, April 2000.

[7] A. K. Dey, D. Salber and G. D. Abowd. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction Journal* 16(2-4), pp. 97-166, 2001.

[8] P. Dockhorn Costa. *Towards a Services Platform for Context-Aware Applications*. Master Thesis. University of Twente, The Netherlands, 2003.

[9] C. Efstratiou, K. Cheverst, N. Davies and A. Friday. An Architecture for the Effective Support of Adaptive Context-Aware Applications. *Proc. 2nd Int. Conf. in Mobile Data Management (MDM'01)*, Hong Kong, pp. 15-26, January 2001.

[10] M. Esler, J. Hightower, T. Anderson, and G. Borriello. Next Century Challenges: Data-Centric Networking for Invisible Computing. *Proc. 5th Annual Intl. Conference on Mobile Computing Networking (MobiCom'99)*, August 1999.

[11] K. Eustice, T. J. Lehman. A. Morales. M. C. Munson, S. Edlund and M. Guillen. A Universal Information Appliance. *IBM Systems Journal*, 38(4), pp. 575-601, 1999.

[12] E. Gamma, R. Helm, R. Johnson and J. Vlissied. *Design Patterns*, Addison Wesley, 1995.

[13] A. Held. Modeling of Context Information for Pervasive Computing Applications. *Proc. 6th World Multiconference on Systemics, Cybernetics and Informatics* (SCI2002), Orlando, FL, July 2002.

[14] J. Indulska, S. W. Loke, A. Rakotonirainy, V. Witana, A. Zaslavski. An Open Architecture for Pervasive Systems. *Proc. 3rd Intl. Work. Conf. on Distributed Applications and Interoperable Systems (DAIS 2001)*, Kraków, Poland, pp. 175-188, 2001.

[15] J. Indulska, R. Robinson, A. Rakotonirainy, and K. Henricksen. Experiences in Using CC/PP in Context-Aware Systems. *Proc. 4th Intl. Conf. on Mobile Data Management*, Melbourne, Australia, pp. 247-261, January 2003.

[16] T. Kindberg and J. Barton. A Web-Based Nomadic Computing System. *Computer Networks*, Elsevier, 35(4), pp. 443-456, 2001.

[17] G. Krasner and S. Pope. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80, *Journal of Object Oriented Programming*, August/September 1988

[18] T. Strang and C. Linnhoff-Popien. Service Interoperability on Context Level in Ubiquitous Computing Environments. *Intl. Conf. on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet (SSGRR2003w)*, L'Aquila, Italy, January 2003.

[19] R. Want, A. Hopper, V. Falcão, and J. Gibbons. The Active Badge location system. *ACM Trans. on Information Systems*, 10(1), pp. 91-102, 1992