Putting Context in Context: The Role and Design of Context Management in a Mobility and Adaptation Enabling Middleware

Marius Mikalsen
Jacqueline Floch
Erlend Stav
SINTEF ICT
Norway
{mariusm,jacf,sta}@sintef.no

Nearchos Paspallis
George A. Papadopoulos
Department of Computer Science
University of Cyprus
{nearchos, george}@cs.ucy.ac.cy

Pedro Antonio Ruiz
Integrasys
Esquillo
Spain
pedro.ruiz@integrasys.es

Abstract

The operating context of mobile applications and services is constantly changing. In order to achieve higher levels of usability, mobile applications and services need to adapt to changes in context. This paper argues the need for adaptation enabling middleware that simplifies the development of context aware adaptive applications, and makes it economically and practically feasible to develop such applications. We claim that the traditional approach of simply providing contextual information to applications and let them handle the adaptation can be ineffective. We suggest a holistic approach where context management is an integral part of a more comprehensive adaptation enabling middleware. This paper describes the role and the design of the context management component in such a middleware architecture. The feasibility of the approach is demonstrated in a scenario where proof-of-concept implementations have been developed and evaluated.

1. Introduction

Smaller computing devices, increasing computing power, and proliferating mobile networks encourage users to bring their computers everywhere they go. This leads to diverse operating conditions, and in order to optimise their usability, systems should be able to adapt to the changing conditions [1], [2]. Although the utility of adaptive and context aware systems has already been demonstrated [3], the ubiquitous computing paradigm suggested by Weiser [4], is yet to come. This is partly due to the high complexity which characterizes the development of such systems, and the lack of generic tool support for the development process [5].

To encourage the proliferation of adaptive and context aware applications and services, cost-effective development methods and tools must be available. This work presents a mobility and adaptation enabling middleware (MADAM) with focus on the context management part of an architecture that aims at providing modelling and tool support for the development of context aware applications. The context manager is a central component, as it is the eyes and ears of the middleware, enabling it to *sense* the operating conditions in the environment, and choose appropriate adaptation strategies.

In the middleware, architecture models are used as the basis for middleware based adaptation [1], [2], [6]. Floch et al [6] argue that middleware based solutions are preferable to other approaches (e.g. using programming features, such as conditional expressions, parameterization, and exceptions), have drawbacks. This is mainly due to the high degree of complexity introduced by the intertwining of the adaptation and the application behaviours. Furthermore, such approaches typically scale poorly when multiple adaptations are triggered. Finally, the inter-mixing of the adaptation and application code renders the software evolution significantly harder. This is inline with Mascolo et al [7].

The MADAM middleware uses architecture models of application variants to reason about, and implement adaptation at runtime. The application variant's metadata provides information about the feasibility of the variant in a given context. The middleware can then find the best application variant for a given context, and implement it (e.g. by reconfiguring the constituent components of the variant). The middleware has three main functions [6]:

- Detect relevant context changes

Proceedings of the 7th International Conference on Mobile Data Management (MDM'06)
0-7695-2526-1/06 $20.00 © 2006 IEEE

- Reason about these changes, and make decisions about which adaptations maximize the utility of the current applications or services
- Implement the adaptation choices

The need for high quality context information is evident as it forms the basis upon which the adaptation component of the middleware can do its reasoning. We argue that having context management as an integral part of such a generic and reusable adaptation enabling middleware is preferable over approaches that require custom context management for their applications, as pointed out by Lei et al in [8]. We further claim that building standalone context management services that deliver context information to applications do not suffice. This claim is in line with the argument for middleware support presented above.

This paper is organised as follows. Section 2 presents the foundations of our work. The research projects we include in this section are chosen to illustrate how our approach builds on established foundations in context research. These works also illustrate how our approach differs, as we see context management as an integral part of a more holistic, adaptation enabling middleware. In section 3 we explain the essence of this middleware approach, where the context manager is one of the three core components. In section 4 we introduce the context manager, and explain its design in light of the requirements put upon it from the surrounding middleware. We also point out how earlier context work relates to this design. Section 5 contains an example that demonstrates our approach, and presents results that justify the approach. In section 6 we discuss our approach in light of related work and explain what directions we see for the future of this research. Finally, in section 7 some closing remarks conclude the paper.

2. Foundations

Our work on the context management component of the adaptation enabling middleware is funded in the research tradition of context-aware computing. Already in 1994, Schilit and Theimer [9] coined the phrase context-aware to describe their work where location, and the identification of nearby people and objects where in focus. Since then, context-aware computing has received a lot of attention covering many topics, beyond the original notion of location. In the following paragraphs we concentrate on three approaches; the *Context Toolkit* by Dey [10], [11] the *Reflective Middleware Solution* by Capra et.al. [12], and finally, the *Adaptive Middleware Framework* by Huebscher et.al. [13].

The Context Toolkit introduced the context widget, which is responsible for acquiring a certain type of context information, and to make this information available to applications. The applications access the widget by using poll and subscribe methods. Context widgets operate independently from the applications that use them. Context widgets facilitate the use of context sensing devices in the architecture, in a way which makes them transparent to the context-aware applications.

The context interpreter provides interpretation functionality, which tries to predict future actions or intentions of the users. The interpreter accepts one or more context entities, and produces a single piece of context. For example, an application can get the context from all widgets in a conference room, and determine that a meeting is taking place. Finally, context aggregators are used to aggregate or collect context information, i.e. they are responsible for all the context of a single entity (such as a person).

Another approach for aggregating context has been developed by Capra et al [12] which use reflection and meta-data to build the system that supports context aware applications. Applications pass metadata to the middleware. This metadata constitutes a policy as to how the applications want the middleware to behave as a result of a specific context occurrence. As context and application needs change continuously, one cannot assume that the metadata are static. Therefore, applications also use the reflection mechanisms offered by the middleware to inspect their own metadata, and possibly alter it according to the changing needs. The metadata format is standardised using XML Schemas.

In a recent work, Huebscher et al [13] present an adaptive middleware framework for context-aware applications. They refer to adaptation as adaptive delivery of context information to context aware applications. Additionally, adaptation is essential for the middleware itself, as it is used to optimise the quality of context information. The middleware adapts itself, but not the applications.

Huebscher et al's middleware framework has 4-tier architecture. The bottom layer consists of sensors, typically physical devices, such as wireless sensor networks, RFID tags etc. These sensors provide raw data. In the next layer, context providers apply sensor logic to produce context types as services. In the third layer, context services retrieve context information from the context providers on behalf of the applications, and deliver this information to the applications. This abstraction layer is important, because if a different context provider is better for an application than the one currently in use, the context service adapts, and switches to the best alternative without having to involve the application. The fourth

and final layer is the application layer. In this layer, the applications are able to select and utilise the best context information available, in order to achieve their context aware behaviour.

3. The mobility and adaptation enabling middleware architecture

The goal of the MADAM project is to provide developers of mobile applications with a middleware platform in support of the development of adaptive applications. Influenced by the work of Szyperski [14], we use component frameworks as a means to build both applications and middleware that are capable of being adapted by reconfiguration. As described by Hallsteinsen et al [1, 2], and by Floch et al [6], components are annotated with properties. Properties are used to qualify the services offered or needed by a component. A user interface may for example, support hands-free operation (offered) if Bluetooth is available (needed). Applications are composed of components that provide particular properties. The middleware platform recognizes and uses these properties in order to reconfigure and adapt the application, in accordance to context changes.

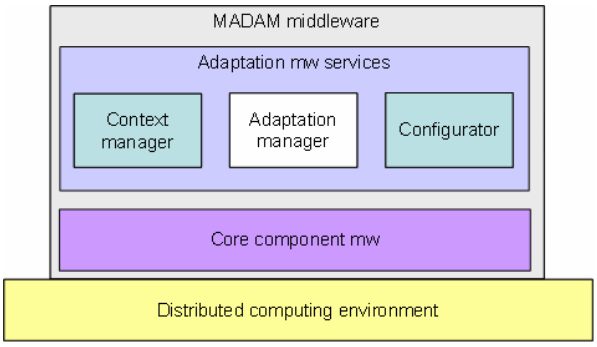

Figure 1 – The MADAM middleware architecture

The overall middleware architecture is shown in Figure 1 above. The core provides the fundamental platform-independent services for the management of applications, components and component instances. The core relies on the basic mechanisms for instantiation, deployment and communication provided by the distributed computing environment.

The adaptation middleware offers three core services:

- The Context manager which monitors the user and the execution context for detection of relevant changes.
- The Adaptation Manager which reasons about the impact of the changes and decides about appropriate adaptations based on architectural description of component properties.
- The Configurator which reconfigures the application variant to put the decided adaptations into effect.

Run-time representations of the application framework and the application instances are available to the MADAM Middleware, enabling adaptation reasoning and reconfiguration. The context model describes the context elements which are relevant for adaptation.

The context manager mainly provides services to the adaptation manager. When an application is started, the adaptation manager analyses the property annotations in the application run-time models in order to identify context information of interest. The adaptation manager then subscribes to relevant context information. New context sensors and reasoners are plugged-in, if necessary. Conversely when an application is terminated, some context information (and sensors) may no longer be needed.

The adaptation manager uses all relevant context information, and tries to find the most suitable application variant. Adaptation will occur in two cases. First, at application start-up, when an initial application configuration is initiated. Second, adaptation will occur when there are sufficient relevant context changes (reconfiguration).

In the following, we introduce the context manager, which is the component that provides the adaptation manager with the contextual information required for adaptation reasoning and reconfiguration.

4. Context management

As we have already stated, our approach to context management extends the research tradition in context awareness. One interesting aspect of the following context management architecture is that it is an integral part of a more holistic adaptation enabling middleware. Therefore, this section provides insight into what features are needed of context managers, in order to fulfill their role in such environments. We start by describing requirements that the surrounding middleware puts on the context manager, and continue by investigating the MADAM context model and context manager architecture to show how they are designed to meet the requirements.

4.1 Context management requirements

In order for the MADAM Middleware to be able to adapt applications, the context manager must meet the following requirements (this list of requirements has

been derived as the result of a scenario-based requirement engineering process, part of the MADAM project):

Define context representations that the middleware can use. Our middleware approach intends to be domain independent. This implies that the context model maintained by the context manager must make no assumptions on what is context in specific domains (e.g. network context might be extremely important in one domain, but less important in another). Instead the context model should define a structure which allows developers to express their domain-dependent context, while at the same time remain applicable to other components of the middleware.

Support context sensing. The context manager should provide support for the definition of context sensors [10, 13, 15] enabling the sensing of information (e.g. user activity, user environment, mobile device resources, availability of various networks, network service quality and cost, and system infrastructure.) Again, since the MADAM platform is domain and platform independent, we can not assume domain specific sensors, and rather design the architecture to allow for domain specific plug-ins

Support context reasoning: In order to reduce context noise, the context manager should allow the creation and deployment of context reasoning mechanisms. Context reasoning can be used to derive higher level context information (e.g. a meeting is taking place) from lower level context information (such as GPS coordinates, or number of persons in the room). It should be easy for other middleware components to specify what particular context information they are interested in, and what changes in context are relevant to them. Other middleware components should not be aware of the internal process in the context manager for obtaining and managing this information. As pointed out by Austaller et al [15], in order to be useful, context management services need to implement strategies for reducing context noise.

These requirements are not exhaustive. Depending on the nature and the goals of the target application, additional features might also be required, to enable, for example, inter-operability between devices, as it is argued in section 6. In the following sections, we show how these requirements have influenced the design of the MADAM context model, and the context manager architecture.

4.2 The MADAM context model

We have adopted Dey's widely used definition of context [10]: "*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 application themselves.*"

The definition intends to be domain independent (as opposed to definitions that include specific context parameters, such as location). The goal of the context model is to provide a generic structure for defining context elements and their properties that are usable by the middleware as a whole.

Building on the work described in [5] and [16], we suggest the simple context model depicted in figure 2. The context information is encompassed in *Context Elements*. The context elements can be composite (i.e. elements within elements). Context elements extend the *Entity of Interest* component. The MADAM middleware conceives a software system and its context to be a set of interacting components. Entities of interest (for the middleware) are then either software entities or context entities.

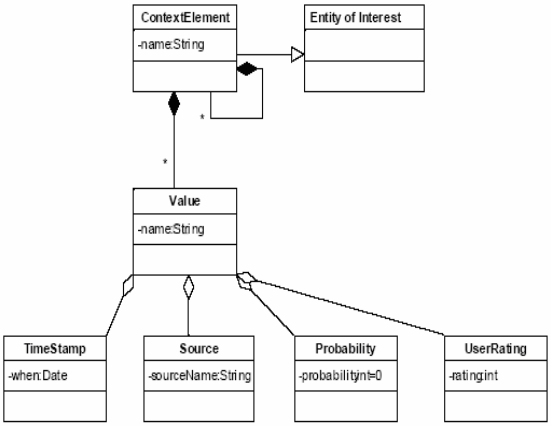

Figure 2 - The MADAM context model

Elements have *Values*. The value is the information available in the element. Every *value* instance encapsulates the actual data. An element can have several values. For example, the WLAN element above can have two values corresponding to bandwidth (e.g. 1Mbit) and cost (e.g. 1 EURO/Mbit)

In their simplest form, a context element consists of an identifier, and a value entry. The context elements must be uniquely identifiable, so that the adaptation manager can map application variants need for context information, to context elements administered by the context manager.

Values are also associated with metadata that functions as Quality of Context parameters. Such metadata is important when performing reasoning on context information. In the model above, four types of metadata are suggested in accordance to MADAM needs; these are the timestamp (when created), source

(who created it), probability (trustworthiness) and user rating (how the user rates the importance of the element). Other research, such as Buckholtz et al [22], suggests to use precision, probability of correctness, trust-worthiness, resolution and up-to-dateness as important Quality of Context parameters.

4.3 Context manager architecture

Our context manager architecture (see figure 3) can conceptually be mapped to the multi-tier architectures of Dey [10, 11] and Huebscher et al [13] in that the raw context data flows into the lower tiers of the architecture (in our case the resource sensors) and are aggregated and reasoned upon up through the layers. In our architecture, we frame aggregation and reasoning tasks in the context sensors and context reasoners.

As suggested by Lei [8], the context manager considers entities that need context information to be context clients, while it considers entities providing context as context sources. In MADAM, the primary client is the adaptation manager, while sources are context sensors and context reasoners. A context source can also be a context client (e.g. a context reasoner which requires context information from other sources in order to aggregate new context information).

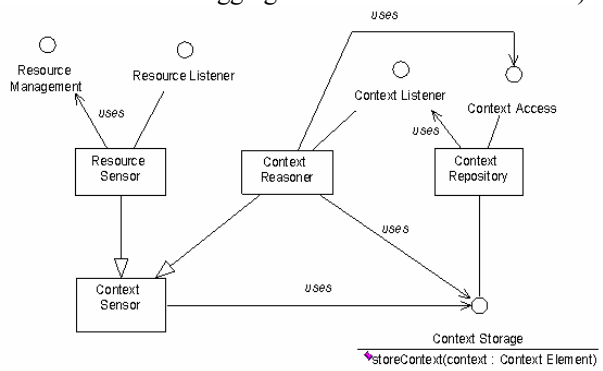

Figure 3 - The context manager architecture

The context manager provides two important interfaces to context clients, the context listener and the context access. The clients can either request to be notified of certain events using the context listener interface (push), or they can use the context access interface to explicitly query context information (pull).

Context repository. The context repository is the main entry point for clients to the context manager. The primary tasks of the context repository are to maintain a context model, register and notify listeners, give access to context elements, and keep registry of available components (sensors, reasoners and storage). In order to get access to a specific context element, a context client (such as the adaptation manager) either registers as an observer to that element, or directly accesses it via the context access interface. These interfaces are similar to the context services described by Huebscher et al [13] and Dey's [10] context aggregators.

Context sensors. The context sensors are components which provide context information to the context repository (a type of context source). Sensors can be wrappers around specialized hardware drivers, or legacy code used for monitoring context, such as battery, memory, and network information.

Context reasoners. Context reasoners extend context sensors, and have additional functionality such that they can produce one or more context elements using other context elements as input. It is an important task for the context manager to reduce context noise [15], by filtering out unnecessary context information, which is not relevant for adapting applications. The adaptation manager should only be notified when a significant change occurs in context. Consequently, the context reasoners need to implement filtering mechanisms. These mechanisms can vary from very simple, rule-based logic, to more advanced techniques. In the MADAM architecture the goal is that reasoners are "plug and play" in order to make it possible to target reasoners according to different needs and domains. For example, the applications can provide the context manager with Quality of Context (QoC) requirements, such as precision and refresh rate, as shown by Huebscher [13]. Another approach is introduced by Kofod-Petersen and Mikalsen [5], where case-based reasoning enabled intelligent agents are used to implement context filtering.

Context storage. Keeping track of historical context information is often required in order to determine trends in context data (for example trends in user behaviour, network stability, etc). The need for storage mechanisms was shown in Dey [10], where context widgets stored all context information they sensed, and in Kofod-Petersen and Mikalsen [5] where a context space, along with a context history abstraction was demonstrated.

5. The janitor scenario

This scenario addresses a janitor company responsible for the maintenance of a variety of technical installations geographically spread. The scenario is relevant as it addresses mobile users that are relying on the mobile terminal in their work, and are experiencing continuous context changes. The scenario therefore addresses a concrete need for adaptation. Several adaptations could occur as a result of context change. First, consider that the janitor is arriving at the

customer's location (sensed using GSM positioning). An appropriate application is launched by the middleware. Second, the janitor is in reach of the customer's WLAN network. This is sensed by the network sensor, and the application's networking component is adapted from GPRS to WLAN, and relevant information is downloaded. Finally, the work has lasted for some time, and the battery level decreases rapidly as a result of WLAN usage. The middleware senses this from the battery sensor, and reconfigures the application to a self-reliant client that only periodically synchronise data with the server.

The above scenario involves all three of the middleware components. First, the context manager notifies the adaptation manager that the janitor is approaching a particular location. The adaptation manager finds the most feasible application variant (component framework) for this context, and let the configurator to start the application. As the application starts, the adaptation manager registers context listeners with the context manager based on the application variant's contextual needs (in the example above battery status and WLAN coverage). Second, the context manager notifies the adaptation manager that WLAN is in range. The adaptation manager knows from the architectural description of the application, that one variant of it has the possibility for WLAN communication. The configurator adapts the application by switching its GPRS network communication component with a WLAN component. Finally, the context manager senses that battery power is decreasing. The adaptation manager finds that the application variant with the highest utility in this case, is one where the application is self-reliant, and where network communication is reduced to periodically synchronising data. The configurator implements the adaptation.

The middleware is implemented as a prototype, and is also used in two industrial pilots. The evaluation so far has demonstrated that the context management provides good support for accessing and identifying context elements, and works well as a manager for context information. The framework supports context dependencies so that only changes in relevant context elements will lead to application reconfiguration. Some of the suggested improvements are that there should be standardised rules in the framework as to how the user needs and preferences influence the relative importance of other context information. There is a clear need for a kind of library of reasoning mechanisms that accept some context information as input, and provides new context information as output. Another finding is that there should always be possible for the user to override context sensing, and explicitly state what is the context. The evaluation has also identified the need for more support for interoperability with third party context sensors. The implications of this are discussed in the next section.

We are currently working to make the prototype implementation and source-code available under an open source middleware initiative such as ObjectWeb (http://www.objectweb.org).

6. Discussion and future work

Several approaches have implemented multi-tier architectures in an effort to provide client applications with abstraction layers, thus freeing them from concerns related to the details of context management. Dey [10, 11] describes a system where context information flows from sensors, through context widgets, and interpreters to context aggregators that gather all context information about an entity (e.g. a person). Lei et al [8] have introduced a similar architecture where dispatchers route application requests to appropriate context drivers. Each context driver handles one type of context information, and encapsulates the details of interaction with context sources (sensors). The same idea is reflected in Huebscher et al [13], where context flows from sensors through context providers and context services, to the client applications. Common for all these approaches is that they do not explicitly address the adaptation of applications, but rather focus on providing the best possible context information to the applications in order for themselves to implement adaptive behavior. Capra et al [12] explicitly state that "*the application… can normally make more efficient and better quality decisions based on application-specific information*". We, on the other hand, believe that providing applications with high quality context information, managing context sensors, aggregating and reasoning on context, is not sufficient. We claim that in addition to these, adaptation enabling middleware should provide support for the adaptation itself (that is; creating the context aware behavior), and that context management realizes its potential better as an integral part of such middleware functionality. We should aim to avoid the complexity introduced when we intertwine core application functionality with adaptive behavior [6].

Middleware support for tasks that are possible to generalize across several applications yields the possibility for optimized middleware implementations. This is comparable to modern database systems. Today, no one would prefer to have application logic implementing the algorithms required for searching within large data-structures. Instead we leave the

responsibility for this task to internal, reusable, and dedicated database mechanisms.

The availability of tool support for standardized tasks (such as the MADAM middleware), also speeds software development, and makes for fewer errors. Consider, for example, the class loader and garbage collector mechanisms of the Java Runtime Environment. Removing such tedious and relatively complex tasks from the software developers contributes to more efficient software development, and fewer runtime errors. Our goal is to provide similar tool support for the development of mobile and context aware applications.

Our results demonstrate that it is feasible to develop applications using the approach described in this work. Several challenges are still ahead and in particular the issue of interoperability. In our middleware we have suggested a context management structure that allows for plugging in domain specific context sensors and reasoners. This gives developers the possibility to use 3rd party context management components (such as Dey's context toolkit). There is however no guarantee that such context management components use context representations that are exploitable by the middleware. We need to investigate strategies that enable middleware frameworks such as MADAM to utilize context from several context management systems. In [18], Carney et al list generic interoperability strategies, which we believe can be applied successfully to this challenge. Two that are particularly promising to context aware computing are the *Semantic Web* and *Service oriented Architectures*. The Semantic Web studies the possibilities of linking up information across the web in such a way such as to be easily process-able by machines on a global scale [20]. To enable such inter-operability, descriptive ontology languages such as *RDF* and *OWL* have been introduced. They act as a means for knowledge sharing between context aware applications. COBRA-ONT is an ontology which was introduced by Chen et al [19] to specifically target context-aware pervasive environments.

SoA has been suggested for use in context aware computing (see for example the Service-Oriented Context-Aware Middleware, SOCAM, by Gu et al [21]). A promising application of these technologies could for example involve the formation of ad-hoc networks between context manager services, enabling sharing of relevant information. SoA in isolation does not accomplish this, as it needs better ways of describing services behavior and the Quality of Service (QoS). Additionally, further work is required to guarantee the semantics of the data used by the service. Although Ontologies and SoA have their limitations, they both appear to be promising approaches for context management in the area of adaptation enabling middleware solutions. Their usefulness is particularly illustrated when we consider collaborative context management, among distributed peers, forming ad-hoc networks.

7. Closing remarks

In this work, we have argued that context management should be an integral part of a holistic middleware approach, thus enabling the development and support of context-aware, adaptive applications. We have argued that in earlier works a lot of focus has been on standalone context management modules, which perform context sensing and aggregation, but leave the responsibility for adaptation to the applications. Unlike these approaches, we have introduced the MADAM middleware and focused on how context management is an integral part of such a holistic approach to adaptation. Furthermore, we have shown how this context management service builds on, and utilizes previous work on context management.

Our vision is to move away from context-aware applications towards partially context-unaware applications, in the sense that applications will be capable of being adapted to context without being aware of the context changes, and the corresponding adaptations per se. Application developers should only need to describe the functional requirements and needs (some of them contextual), and leave context sensing, context reasoning, as well as most of the adaptation to middleware tools. We expect this to be the natural evolutionary step. As the mobile and context-aware computing paradigm matures, middleware tools will support standardized tasks, much the same way databases perform standard data management today. We have shown that such an approach is feasible, by developing and evaluating two pilot applications. Experimenting with the approach, we have identified ontologies and service oriented architectures as relevant approaches in relation to adaptation enabling middleware.

8. Acknowledgements

This work was funded by the EU commission as part of the IST MADAM project (6th framework programme, contract number 4169). http://www.ist-madam.org/

9. References

[1] Hallsteinsen, S., Stav, E., and Floch, J. Self-Adaptation for Everyday Systems. In Proceedings of ACM SIGSOFT

Workshop on Self-Managed Systems(WOSS '04), Oct 31-Nov 1, 2004 Newport Beach, CA, USA

[2] Hallsteinsen, S., Floch, J., and Stav, E. A Middleware Centric Approach to Building Self-Adapting Systems. In Proceedings of Software Engineering and Middleware (SEM 2004) 20-21 September 2004, Linz, Austria.

[3] Bristow, H. W., Baber, C., Cross, J. and Wooley, S. (2002) Evaluating contextual information for wearable computing. In Proceedings of the Sixth International Symposium on Wearable Computers, New York: IEEE Computer Society.

[4] Weiser, M.: Some computer science issues in ubiquitous computing. Communications of the ACM 36 (1993)

[5] Kofod-Petersen, A. and Mikalsen, M.(2005) Representing and Reasoning about Context in a Mobile Environment. Revue d'Intelligence Artificielle (RIA), vol 19, no. 3, pp 479-498.

[6] Floch, J. Hallsteinsen, S. Stav, E. Eliassen, F. Lund, K. Gjørven, E. "Beyond design time: using architecture models for runtime adaptability" To appear in: IEEE Software Volume 23, Issue No. 2. 2006.

[7] Mascolo, C., L. Capra, and W. Emmerich. "Mobile computing middleware" in Advanced Lectures on Networking. NETWORKING 2002 Tutorials. 19 24 May 2002 Pisa, Italy. 2002: Springer-Verlag, Berlin, Germany.

[8] Lei H., Sow D. M., Davis J. S. I., Banavar G., Ebling M. R., "The design and applications of a context service", Mobile Computing and Communications Review, 6(4): 44–55, 2002. ACM SIGMOBILE.

[9] Schilit B. N., Theimer M. M., "Disseminating active map information to mobile hosts", *IEEE Network*, 8(5):22–32, September/October 1994.

[10] Dey A. K., Providing Architectural Support for Building Context-Aware Applications, PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.

[11] Dey A. K., "Understanding and using context.", Personal and Ubiquitous Computing, 5(1):4–7, 2001.

[12] Capra L., Emmerich W., Mascolo C., "Reflective middleware solutions for contextaware applications", In A. Yonezawa and S. Matsuoka, editors, Metalevel Architectures and Separation of Crosscutting Concerns - Proc. of Reflection 2001, pages 126–133. Springer Verlag.

[13] Huebscher, M. C., McCann, J. A. An adaptive middleware framework for context-aware applications, Personal and Ubiquitous Computing, Volume 10, Issue 1, Feb 2006, Pages 12 – 20.

[14] Szyperski, C., "Component Software: Beyond Object-Oriented Programming", Addison Wesley, 1997 (2nd ed. 2002, ISBN 0-201-74572-0).

[15] Austaller, G., J. Kangasharju, et al. (2004). Using Web Services to build Context-Aware Applications in Ubiquitous Computing. Web Engineering: 4th International Conference (ICWE 2004). Munich, Germany, Lecture Notes in Computer Science.

[16] Göker, A., Myrhaug, H.I.: User context and personalisation. In: Workshop proceedings for the 6th European Conference on Case Based Reasoning (2002).

[17] Garlan, D., et al., *Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure.* Computer, 2004. 37(10): p. 46-54.

[18] Carney, D., Fisher, D., Morris, E., Place, P. Some current approaches to Interoperability. Carnegie Mellon University Technical Note. CMU/SEI-2005-TN-033. 2005.

[19] Chen, H., Finin, T., Joshi, A. "An ontology for context-aware pervasive computing environments" The Knowledge Enginnering Review. Cambridge University Press. Vol 18:3. pp 197-207. 2004.

[20] Asunción Gómez-Pérez, Jérôme Euzenat, *Editors*, 2nd European Semantic Web Conference (ESWC2005), Springer Verlag LNCS Vol. 3532, Heraclion, Greece, 2005.

[21] Gu, T., Pung, H. K., and Zhang, D. Q. 2005. A service-oriented middleware for building context-aware services. *Journal of Network and Computer. Appications.* 28, 1 (Jan. 2005), 1-18.

[22] T. Buchholz, A. K¨upper, and M. Schiffers. Quality of Context: What It Is and Why We Need It. In *10th Workshop of the HP OpenView University Association (HPOVUA'03)*, Geneva, Switzerland, July 2003.