

Evolutionary and Efficient Context Management in Heterogeneous Environments*

Ricardo Couto A. da Rocha[†], Markus Endler
Department of Informatics
Pontifícia Universidade Católica do Rio de Janeiro
R. Marquês de São Vicente, 225
22453-900 - Rio de Janeiro, Brazil
{rcarocho, endler}@inf.puc-rio.br

ABSTRACT

Mobile computing and pervasive environments are mainly characterized by heterogeneity of devices, with different capabilities, resources, operating systems and applications. In a realistic scenario for context-aware computing, middleware should be deployable in the whole distributed system, despite device's resource limitations, and the developer should be able to evolve the context model when new context-aware applications or context providers are introduced. This paper discusses how context modeling and design of middleware architecture can impact on the efficiency of provision, distribution and access of context information in heterogeneous environments. This paper describes a middleware architecture and design strategies in order to address such requirements.

Categories and Subject Descriptors

C.2.1 [Computer-Communication Networks]: Network Architecture and Design—*Wireless Communications*; C.2.4 [Computer-Communication Networks]: Distributed Systems—*Distributed applications*

General Terms

Design, Performance, Management

Keywords

Middleware, Middleware design, Context-awareness, Context modeling, Context management

*This work is being funded by CNPq Research Grant no. 479824/2004-5.

[†]Supported by CNPq.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MPAC'2005 November 28- December 2, Grenoble, France
Copyright 2005 ACM 1-59593-268-2/05/11 ...\$5.00.

1. INTRODUCTION

Context-aware computing is widely accepted as a suitable paradigm for the development of applications for mobile and pervasive environments. The main purpose of context-aware computing is to allow the dynamic adaptation of applications and services in order to guarantee an adequate usage of device and network resources, and to properly handle runtime requirements of applications. Context-aware middleware should ideally enable specification and management of adaptations of three types: service, inter-services and application [2].

One of the main characteristics of mobile computing environments is its heterogeneity: the environment is composed of devices using different technologies, resources, operating systems and network capabilities. Such heterogeneity produces an important impact in middleware development. From a practical viewpoint, middleware should be deployable in devices with different resource limitations [15]. Moreover, heterogeneity must be supported without compromising other important requirements for distributed systems, like performance, availability and security.

Evolution is another aspect directly related to heterogeneity. In environments supporting context-aware applications, the modeled context can change when new applications, sensor or context interpreters are introduced. Consequently, context models should be flexible in order to accommodate complex context structures and relationships, as well middleware should ease incorporation of new context types at runtime.

This paper discusses the impact of heterogeneity in context models and middleware development, and presents some policies for context information access that can improve overall context-aware middleware performance. We present an architecture that takes into account the heterogeneity and the context model to allow efficient interaction among devices and services - mainly for the purpose of context information access - and how it contributes to an evolutionary perspective of context usage.

The remainder of this paper is organized as follows. Section 2 discusses how context modeling and middleware architecture design can be affected by heterogeneity concerns. Section 3 discusses some policies for context access and exemplifies how they are adopted in the current middleware research. Section 4 presents an middleware architecture and policies we are proposing to address the challenges of han-

dling context in heterogeneous environments. Finally, section 5 presents the current stage and future directions of our research and some conclusions of this paper.

2. CONTEXT IN HETEROGENEOUS ENVIRONMENTS

Research in context-aware computing has been exploited in two main directions: context modeling and middleware support. Research in context modeling has been focusing on development of context models that accounts for the variety of mobile and pervasive environment requirements. Research in middleware for context-aware systems has been oriented towards the development of middleware architectures that support many adaptation paradigms and performance guarantees, despite of resource limited devices. These research areas are directly connected, since the context model adopted by a middleware may require the adoption of more complex context management and storage mechanisms. In other words, the complexity of a context model determines the complexity of a middleware context handling capability. Coutaz *et al* [5] presents this relationship as a conceptual framework that interconnects an ontological foundation for context modeling with a runtime infrastructure (middleware). We believe that such interdependence has been neglected in most middleware projects. As result, most middlewares have adopted context models that are restricted to a single context domain (e.g. local execution context [2, 4, 11, 21, 22] or location [16]), or, alternatively, the complexity of the middleware’s context processing hinders its deployment in resource limited devices.

The choice of the context model impacts the middleware development in three ways: (a) the context model may increase the complexity of infrastructures required to handle context; (b) a middleware can use context modeling to infer how context will be used by applications; for example, a modeling could specify if the use of a context is limited to the device that produced it; and (c) the context model can allow applications to set precision requirements that may affect performance of context dissemination. For example, context models that support quality-of-context can define several degrees of precision and freshness and determine how middleware is supposed to deliver the context. The analysis of these interrelationships could open opportunities for performance enhancement in middleware.

Another aspect to be considered is evolution of the context models: on one hand a context model should be flexible and generic to allow inclusion of new context information, but on the other hand middleware architectures should efficiently handle new context.

We are mainly interested in the challenges that heterogeneity imposes on context modeling and middleware design. In pervasive environments, heterogeneity has hardware (physical), software (logical) and architectural aspects. *Hardware heterogeneity* is characterized by the presence of different computing devices, such as desktop computers, PDAs and mobile phones, as well as different network technologies integrating these devices. The challenging consequence of such heterogeneity is its demand for middleware infrastructures that can be deployed both on server workstations and on portable mobile devices. *Software heterogeneity* is characterized by the presence of different operating systems and applications, which demand software inter-

Aspect	Characteristics	Impact
Hardware	- resource limitations - network interfaces	MW
Software	- applications - local context - scope of context	M/MW
Architecture	- context domains - network architecture	M/MW

Table 1: Impact of Heterogeneity Aspects in Design of Pervasive Systems

operability and the adoption of context models addressing specific application requirements. *Architectural heterogeneity* is achieved when network interconnections among devices do not respect any known architecture. Hence, such heterogeneity demands adaptable and scalable middlewares in order to provide seamless communication. It also suggests the context models may be limited to some network domains, because some context information may only be relevant to a set of applications or devices. Table 1 summarizes the impact of such heterogeneity aspects in context models (M) and middleware development (MW). In addition, each of these aspects has an evolutionary component: when a new device type or application is introduced in a environment, context models and software infra-structure should be adapted, in order to accommodate the new requirements.

Some previous research has already focused at handling heterogeneity of pervasive environment. For example, CoCo [1] handles heterogeneity of context information services, caused by the adoption of different context models and context services in different network domains. CoCo provides an infrastructure and a language to describe context models and provide interoperability among context information services. However, we are not interested in this kind of heterogeneity. Hence, we assume that any context service and infrastructure is built upon a same middleware.

The following sections discuss how context modeling and middleware design can address heterogeneity issues.

2.1 Context Models

According to Strang and Linnhoff-Popien [23], ontology-based and object-oriented based models are the only models suitable for context modeling for pervasive computing [23].

Several projects have adopted the object-oriented (OO) paradigm to model context information, such as GUIDE [4]. This modelling approach is simple, easy to deploy and efficient, since OO languages are widely adopted. However, the use of an OO context model requires implementation of infrastructures to support all context operations, such as storage and querying.

Most of current research in context modeling focuses the use of ontologies to specify context information and the relationships among contexts (e.g. [3, 18, 17, 24]). Ontologies provide a powerful paradigm for context modeling that offers rich expressiveness and support the evolutionary aspect of context modeling. In order to process ontologies, the system architecture must provide an ontology engine capable of making inferences over ontologies. This requirement may impacts on the performance of local context management in resource limited devices because they are imposed to maintain the ontology engine at a server, instead of locally. Some

middlewarees [3] use agents to transfer the resource-hungry computing for handling ontologies to servers at wired network.

2.2 Middleware Support

Most middleware research efforts have handled heterogeneity in mobile and pervasive systems either as an interoperability issue or "solving it at a higher level, using static and dynamic reconfiguration" [12]. However, one orthogonal question that remains is: What policies, strategies and architectural approaches can improve middleware overall performance and scalability? Roughly speaking, currently proposed middlewarees try to satisfy such requirement by offering performance enhancing infrastructures, distribution management and suitable paradigms for context-aware computing.

Since early work on context-aware middlewarees, there is a common effort to allow the inclusion of context sensors and providers of different technologies and as an evolution of the whole system. For example, Context Toolkit [21] offers the abstraction of widget, which applies for context providers and context interpreters. Context Information Service [14] proposes a similar approach in order to provider context handling in a more generalized way.

Another direct opportunity for performance and scalability enhancements is the context dissemination by distributed services [10], instead of centralized context services as [6]. However, some centralized services can still be used to disseminate context for specific domain of context interest. For example, SCI middleware [7] splits the mobile environment into *ranges*, which define domains of devices and applications where context information is managed by a single context server.

SCI's light-weight software component called *context entity* [7] allows the deployment of resource limited devices or sensors in its architecture. Middlewarees that adopt ontology-based context modeling [17, 8] usually adopt an agent-based interaction paradigm allowing resource limited devices to process context-based computations. However, this approach can increase the number of interactions over wireless links and limit the use of context when the device is running disconnected from network.

Another opportunity for performance enhanced is the adoption of suitable communication paradigms for mobile systems which are better suited to the use of wireless links and intermittent connectivity. Several middlewarees have adopted publish/subscribe [20] and tuplespaces-based [10] communication approaches.

In order to support an evolutionary use of context, some middlewarees [25] explore the concept of context discovery. A middleware that support this concept allows applications to discover and use new context types and sources at runtime.

3. ASPECTS OF CONTEXT ACCESS

Heterogeneity creates some complex scenarios for context access. Figure 1 illustrates a scenario in which parts of a context information are provided and maintained by distributed entities.

This scenario shows that, in heterogeneous environments, contexts tends to be highly distributed and that evaluation of context attributes imposes network communication through wireless and wired links.

Middlewarees for context-aware computing usually adopt

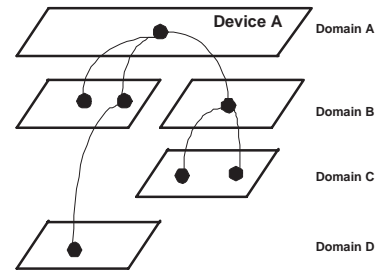


Figure 1: Example of Context Distribution

some implicit policies or restrictions to context access which have direct impact on performance of context management. We have identified six aspects: *communication*, *distribution*, *updating*, *context service placement*, *context evaluation*, and *context querying*.

Communication. The communication policy specifies how context information will be disseminated or gathered through the network by interested applications. *Asynchronous* communication has been widely accepted as the suitable communication paradigm for mobile computing. Middlewarees commonly implement this communication approach adopting either a publish/subscribe-based paradigm [20] or a tuple-spaces-based paradigm [10]. However, *synchronous* communication is still important for context-aware computing as it allows querying context data. For example, in a location-aware system, an application may need to query the set of devices located in a specific area.

Distribution. The distribution policy specifies how access to distributed context information is supported. Some middlewarees [2] offer only support for *local context*, i.e. context data probed and used only at a device. Current middlewarees take a more flexible approach and support *distributed contexts* that can express situation of a set of entities that may be distributed over a network.

Updating. The updating policy specifies how a context information is updated at a repository and how this update is disseminate to context consumers. The most common policy is to send *frequent updates* in order to guarantee that context consumers access the up-to-date context information. Another approach is to enable context consumers to use *intelligent components* that infer the actual information based on a history, or evaluate the information's freshness through a comparison of timestamps. This approach is very useful to manage quality-of-context attributes, such as information confidence.

Context Service Placement. The placement policy specifies where the context storage and processing will be hosted. The simpler policy is the adoption of a *centralized* context service [21] but it imposes performance and scalability problems. *Distributed* context services can be implemented using four approaches: interconnecting context services distributed through a network [7], maintaining a context service in order to handle local context plus a remote service to disseminate context among other devices, implementing ad hoc services to enable context management in ad hoc net-

works [26], and using a combination of the aforementioned solutions.

Context Evaluation. The context evaluation policy specifies how a context-aware application evaluates each attribute that composes a context information. There are two approaches: *eager* evaluation and *lazy* (on-demand) evaluation. The performance of each approach depends on how complex is the context model and the behavior of the application that uses the context.

Context Querying. The context querying policy specifies if the middleware supports context querying, and how it is executed over context repositories. For some middleware, context querying is *unavailable*: context consumers should register interest in context in order to be notified asynchronously when it changes. Consumers infer that the current context state is the last received notification. Middlewares can support queries in two means: *centralized handled queries*, in which a query is sent directly to the repository that maintain the context, and *distributed queries* [9], in which a query can be distributed among several repositories and the result is composed as a unique context information.

4. ARCHITECTURE OF A CONTEXT SERVICE

In order to address such challenges, we have designed a Context Service for the MoCA middleware [20]. This context service aims at flexibility, allowing runtime incorporation of new context information types and new context inference agents. We designed both the context service architecture and the context modeling approach, as they are closely related.

The design of our context service was driven by the following requirements: (1) adoption of a generic and flexible context model that could be dynamically deployed in the middleware architecture; (2) improved performance, allowing the fast access to and dissemination of context information, avoiding the intensive use of network and memory in limited-resource devices; and, (3) support for interoperability, facilitating the use of the service on different devices, operating systems and programming languages.

In our approach, essentially three components interact to create, disseminate and use context information: *context provider*, *context consumer* and *context service*. Each of these components are network entities, represented by applications, services or hosts. The context provider is an entity responsible for publishing a certain context information; it can be the generator of a raw-sensed data or just an interface between a sensor and the middleware. The context consumer is an entity interested in a given context information. The context service is responsible for receiving and storing context information, and to disseminate it to context consumers.

Unlike some architectures that suggest the use of specialized context providers (e.g. aggregators in [21]), MoCA's Context Service adopts a homogeneous approach, in which any context provider uses a single API and the same infrastructure. A context provider can publish either device's local context, that would serve only for the purpose of the hosting device, or context of a less-specific domain which can be disseminated by hosts distributed over a network. A

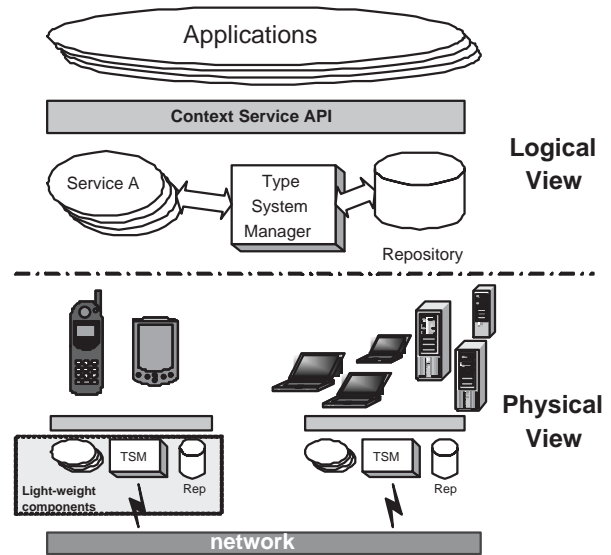


Figure 2: Context Service Architecture

context provider can even be a context inference agent, such as MoCA's LIS service [13] which transforms IEEE 802.11 RSSI values into location information.

Instead of defining specialized context providers, our architecture offers a middleware implementation according to the target device's resource limitations. In resource limited devices, such as PDAs and mobile phones, the middleware actual implementation adopts policies and services in order to decrease the use of network resources, memory usage and access context efficiently. For example, for this sort of device, the context service is configured in a way to keep the view of current local context as simple as possible and avoid using resource hungry services such as a log. Some of these properties are defined statically, when the middleware is deployed at the device, and some are dynamic. Depending on the requirements of the consumers and providers of a device's local context, a negotiation among clients and the middleware shall configure the service as best suited as possible to the local application requirements. Of course, there is a trade-off between the properties that can be dynamically negotiated and the complexity of the middleware implementation, what may interfere in the resource consumption as well. Section 4.3 presents our approach for static and dynamic middleware configuration.

4.1 Middleware Architecture

The *Context Service* is composed of the basic elements shown in figure 2. These elements have the following responsibilities:

Context API a layer that provides to application a unified access to any middleware service.

Event Service is responsible for providing asynchronous communication, delivering contexts and events to interested clients. The event services adopts a publish/subscribe paradigm and offers a specialized API to handle subscriptions for contextual events.

Type System Manager maintains the dynamical context type system, solving and recognizing the available con-

text types at runtime. The type system manager also solves context domains and its location, when a context is composed of sub-contexts placed in different network locations.

Repository maintains the database of context data.

Sub-services additional services that ease the development of client applications, and improves the performance of the whole system. Cache and query services are examples of such sub-services. The query service is responsible for translating client queries to context repository queries and deliver its result to the client.

The behavior and internal architecture of these components may change depending on the device at which the middleware instance is deployed. The overall system is composed of several distributed middleware instances that interact with each other.

In order to make adequate use of device resources and improve the overall system performance, we propose that a specific version of the above components be created for each device class. For example, server hosts have enough resource to store the all context types as well as maintain a context repository that stores a huge context database, offering services such as context history that demands a large amount of disk space. In contrast, in a resource limited device, the type system usually only maintains local context types and a volatile context repository. The device's operating system may also influence the middleware behavior: for example, in non-multithreaded OS (e.g. Palm OS), the middleware will act passively, adopting different policies for context provisioning.

This complexity for middleware behavior selection suggests to keep the interaction among components as simple as possible. In order to address such simplicity, complementary services are implemented as loosely-coupled services. One example is our privacy service [19], that is implemented as an orthogonal service of MoCA's architecture, differently of other approaches such as [11].

4.2 Deployment of Context Types

One of the fundamentals of our approach is strong typed context handling. A context model is mapped to object-oriented language constructions that the developer uses in his application in order to access context information. Application-specific context is handled using the same approach.

We follows an OO model for context handling, instead of an ontology-based model, because, currently, limited resource devices do not allow the usage of local ontology engines.

Figure 3 shows the steps for deployment of new context type. The deployment of a new type occurs when a new context must be introduced in the context-aware system. There are two main steps: context modeling and the context model processing.

The first step consists of modeling the new context information using our XML-based modeling approach. In this XML file, the context modeler specifies attributes, characteristics, relationships with previously specified context, quality-of-context attributes and the provided queries to request context synchronously. A detailed discussion about our context model is not in the scope of this paper.

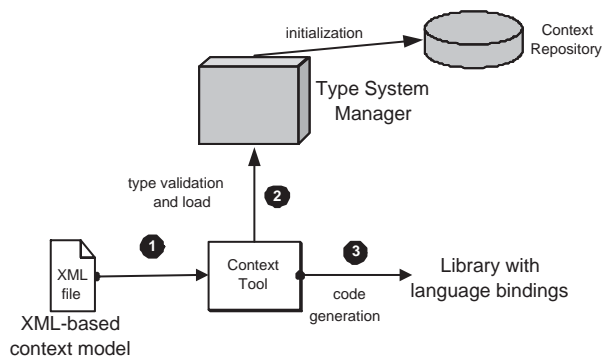


Figure 3: Context Deployment Steps

In the context model processing step, a *Context Tool* reads the XML file and executes the following tasks:

1. Validates the XML syntax and the context model;
2. Updates the context type system and initializes the repository for storing the new context information;
3. Generates a library containing the language bindings to the deployed context, similarly to the RSCM's approach [26]. Currently, we have only implemented a Java language bindings for context information.

When an application developer needs to use a context information, he reads the context's XML modeling file to understand the context semantics, and includes the binding library to his application. The language bindings allows the application to access context information as object references or attributes, and query contexts using object methods. Dependencies with another context information are included in the binding file. To start using a context, applications must be registered at the middleware, identifying its requirements for context provisioning, which will be discussed in next section.

4.3 Configuration of Context Access

Another important issue for improving the performance of context handling is context evaluation, i.e. how a context information obtained from the middleware is evaluated at runtime by applications. We have adopted an approach of setting the middleware to use adequate policies for context access, depending on the context models, application requirements and context domains.

A context domain is a logical boundary that establish the scope of a context information. At least two context domains should be investigated when designing middleware: local domains and non-local domains. A local context domain contains context information provided by a device, such as cpu usage, available memory and operating system. On the other hand, non-local context domain contains context information provided and maintained from another context provider, so an access to its context requires, at least, one hop through the network. Our middleware provides a same API for accessing context of both local and non-local context domains, so application does not need knowledge about middleware behavior or context information distribution in order to implement efficient access.

In order to access and evaluate context information efficiently, we adopted a *static* and a *dynamic* middleware configuration that adapts context evaluation behavior to the properties of the context modeled and to runtime application requirements.

4.3.1 Static Configuration

In order to access efficiently context of local domains, our middleware implements a light-weight local instance of a context type system and context repository, instead of keep any local context in a remote context repository. The middleware provides a local cache service that improves access performance.

Additionally, the middleware behavior changes depending on how a context information was modeled. At context deployment time, the *Context Tool* verifies the attributes of a context information and sends to the context service information that allows it to choose the more suitable policy for context access. For example, consider a context attribute declared as static, i.e. an attribute that has a constant value (e.g. the OS type/version used by a device). When deploying this context in a context type system, the context service is configured to disseminate and update this attribute only at first time an application requests the context.

Our context model specifies some other properties for context attributes that help middleware to process context more efficiently. Such semantic properties can help application developers to understand the dynamic behavior of a context information and, in some cases, to develop applications that use a context information more adequately.

4.3.2 Dynamic Configuration

Besides static configuration based on context model, our middleware allows dynamic configuration **at application startup time**. When an application is started and registers itself at the middleware, it can define specific policies for using a context information. These policies are based on application requirements about the precision of context information it is interested in.

Freshness and *on-demand* (lazy) context evaluation are examples of access policies that can be set by application. Several applications using a same middleware instance can select a different policy for accessing a same context information. In this case, the middleware chooses the more restrictive policy in order to adhere all application requirements. Using such dynamic configuration, the middleware can choose the better moment to publish context and execute context queries.

We are investigating how quality-of-context modeling can be used to provide a richer dynamic middleware configuration. Currently, we are just planning to implement freshness and context precision properties as parameters used to setup our middleware policies.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have discussed the mutual dependence between context models and the design of context-provisioning middleware, and have highlighted six core aspects related to the implementation of middlewares for distributed, heterogeneous and evolutionary context management. Focusing on context evolution, device heterogeneity and efficient context distribution in a mobile network, we have proposed a flexible architecture of a context service

that allows for the dynamic addition and removal of new types of context information. This architecture has means of describing the interrelationship among context types and also provides support for the development and maintenance of context-aware applications.

So far, we have developed a prototype of a context service based on our architecture, and which is now being integrated with other components and services of our context-provisioning middleware architecture MoCA, in particular, with our privacy service for context access. Through this integration, we aim to validate the architecture's ability to support configuration of context access and evolution of the context model.

As part of our future work we will investigate extensions to our architecture in order to accommodate also quality-of-context parameters in a flexible way, and explore dynamic configuration of the middleware at changes of the context types and access requirements. We are also planning to research how context consumers can specify and use *context views*, i.e. selected portions (attributes) of a complex context type. We believe that specifying a context view without changing a context model can create new opportunities for enhancing the efficiency of context access.

6. REFERENCES

- [1] T. Buchholz, M. Krause, C. Linnhoff-Popien, and M. Schiffers. CoCo: Dynamic composition of context information. In *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)*, pages 335–343, 2004.
- [2] A. T. S. Chan and S.-N. Chuang. MobiPADS: a reflective middleware for context-aware mobile computing. *IEEE Transactions on Software Engineering*, 29(12):1072–1085, Dec 2003.
- [3] H. Chen, T. W. Finin, and A. Joshi. Using OWL in a pervasive computing broker. In *Workshop on Ontologies in Open Agent Systems (OAS)*, pages 9–16, Melbourne, Australia, July 2003.
- [4] K. Cheverst, K. Mitchell, and N. Davies. Design of an object model for a context sensitive tourist GUIDE. *Computers & Graphics*, 6(23):883–891, 1999.
- [5] J. Coutaz, J. L. Crowley, S. Dobson, and D. Garlan. Context is key. *Commun. ACM*, 48(3):49–53, 2005.
- [6] A. K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.
- [7] R. Glassey, G. Stevenson, M. Richmond, and P. Nixon. Towards a middleware for generalised context management. In *First International Workshop on Middleware for Pervasive and Ad Hoc Computing, Middleware 2003*, 2003.
- [8] H. Harroud, M. Khedr, and A. Karmouch. Building policy-based context aware applications for mobile environments. *Lecture Notes in Computer Science*, 3284:48–61, Jan. 2004.
- [9] J. Heer, A. Newberger, C. Beckmann, and J. I. Hong. liquid: Context-aware distributed queries. *Lecture Notes in Computer Science*, 2864:140–148, Jan. 2003.
- [10] J. I. Hong and J. A. Landay. An infrastructure approach to context-aware computing. *Human-Computer Interaction*, 16(2, 3 & 4):287–303,

- 2001.
- [11] H. Lei, D. M. Sow, I. John S. Davis, G. Banavar, and M. R. Ebling. The design and applications of a context service. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):45–55, 2002.
- [12] C. Mascolo, L. Capra, and W. Emmerich. *Middleware for Communications*, chapter Principles of Mobile Computing Middleware, pages 261–280. John Wiley and Sons, 2004.
- [13] F. N. d. C. Nascimento. A service for location inference of mobile devices based on IEEE 802.11. Master's thesis, Departamento de Informática, PUC-Rio, August 2005. (in portuguese).
- [14] J. Pascoe. Adding generic contextual capabilities to wearable computers. In *ISWC '98: Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, page 92, Washington, DC, USA, 1998. IEEE Computer Society.
- [15] K. Raatikainen, H. B. Christensen, and T. Nakajima. Application requirements for middleware for mobile and pervasive systems. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):16–24, 2002.
- [16] A. Ranganathan, J. Al-Muhtadi, S. Chetan, R. Campbell, and D. Mickunas. MiddleWhere: a middleware for location awareness in ubiquitous computing applications. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 397–416, New York, NY, USA, 2004. Springer-Verlag New York, Inc.
- [17] G. Rey and J. Coutaz. Contextor: capture and dynamic distribution of contextual information. In *Proceedings of the 1st French-speaking conference on Mobility and ubiquity computing*, pages 131–138, New York, NY, USA, 2004. ACM Press.
- [18] G.-C. Roman, C. Julien, and Q. Huang. Network abstractions for context-aware mobile computing. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 363–373, New York, NY, USA, 2002. ACM Press.
- [19] V. Sacramento, M. Endler, and F. N. Nascimento. A privacy service for context-aware mobile services. In *SecureComm '01: Proc. of the First IEEE/CreatNet International Conference on Security and Privacy for Emerging Areas in Communication Networks*, September 2005.
- [20] V. Sacramento, M. Endler, H. K. Rubinsztein, L. S. Lima, K. Goncalves, and F. N. do Nascimento. MoCA: A middleware for developing collaborative applications for mobile users. *IEEE Distributed Systems Online*, 5(10), Oct. 2004.
- [21] D. Salber, A. K. Dey, and G. D. Abowd. The Context Toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM Press.
- [22] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, USA, December 1994.
- [23] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *First International Workshop on Advanced Context Modelling, Reasoning And Management*, Nottingham, England, Sept. 2004.
- [24] T. Strang, C. Linnhoff-Popien, and K. Frank. CoOL: A Context Ontology Language to enable Contextual Interoperability. In J.-B. Stefani, I. Dameure, and D. Hagimont, editors, *LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003)*, volume 2893 of *Lecture Notes in Computer Science (LNCS)*, pages 236–247, Paris/France, November 2003. Springer Verlag.
- [25] G. Thomson, M. Richmond, S. Terzis, and P. Nixon. An approach to dynamic context discovery and composition. In *Proceedings of UbiSys '03, System Support for Ubiquitous Computing Workshop*, Seattle, Washington, USA, October 2003. UbiComp 2003, The Fifth Annual Conference on Ubiquitous Computing.
- [26] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. S. Gupta. Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing*, 1(3):33–40, 2002.