Middleware for Ubiquitous Context-Awareness*

Ricardo Couto A. da Rocha† , Markus Endler, Thiago Senador de Siqueira
Department of Informatics
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)
R. Marquês de São Vicente, 225
22453-900 - Rio de Janeiro, Brazil
{rcarocha, endler, thiago}@lac.inf.puc-rio.br

ABSTRACT

This position paper discusses the challenges and trade-offs of implementing a middleware that supports ubiquitous context-awareness, i.e., a scenario where context-aware applications may move throught network environments without suffering disruptions in their context-based interactions. We present a middleware approach based on the concept of context domains that satisfy some of the requirements of this scenario.

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*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*

General Terms

Design, Performance, Reliability

Keywords

Middleware, Context-awareness, Ubiquitous Computing, Context management

1. INTRODUCTION

Most middleware platforms for context-awareness are application-oriented and restricted to specific scenarios, such as smart meeting/classrooms [4, 16], smart touristic guides, and web content adaptation. Hence, these platforms either work only for certain environments, or at best, let the mobile clients experience disruptions of context access when they migrate between different environments, be they logical or physical environments. Thus, currently environments enabling some form of context-awareness are isolated and independent, and do not support any application-transparent handover among each other. We call these isolated environments *context-aware islands*, because they hinder the implementation of applications with cross-environmental interest in context information, i.e., the context-aware interest of an application is a combination of contextual situations provided by context sources of different environments.

Some middleware systems support such a scenario by offering distributed platforms for context management [12, 10, 8, 3], federations of context-aware systems [7, 14, 2], or through bridges [11] between context-aware systems. However, we argue that such middlewares do not provide effective solutions, because their generality or scalability is limited. On one hand, more generic and flexible solutions usually only work properly with a small client base, since they impose high complexity on the context management mechanism. On the other hand, solutions that scale with the number of clients often support only lower-level abstractions for context usage for a specific application domain.

We characterize this challenge as the development of a middleware that supports ubiquitous context-awareness. *Ubiquitous context-awareness* is thus the capability of a context-aware system or middleware to provide anytime access to heterogeneous, distributed, and un-anticipated context information in global scale and for distinct scenarios, allowing clients to discover new context types and information, restricted to a specific environment, and without compromising semantic context interoperability. In order to support this scenario, research in middleware must tackle issues such as: (context) generality, extensibility, computing and communication costs, portability, evolution support, and scalability [5]. There is a trade-off between these aspects and, unfortunately, there is not a single solution that adequately satisfies all these requirements.

This paper discusses the challenges, requirements and trade-offs of developing a middleware that provides ubiquitous context-awareness. We present an approach for context management that deals with some of these challenges and discuss some of our design decisions.

2. UBIQUITOUS CONTEXT-AWARENESS

Ubiquitous context-awareness is the ability to dynamically discover *context-aware computational environments* associated with the physical environment, and to keep both the access to context sources and the feasibility of the corre-

*Supported by the CNPq-BMBF Brazilian-German Collaboration Programme (CNPq grant 490817/2006-8) and CNPq grant 474188/2007-8.

†Supported by CNPq.

MPAC'08 December 1-5, 2008 Leuven, Belgium

sponding context-based adaptations, despite client mobility and environment heterogeneity. A *context-aware computational environment* is a logical domain that comprises context types and their instances (e.g. context data), that are limited to this domain.

In order to provide ubiquitous access to context instances, a context-aware system must provide an infrastructure for efficient access to context data in a large-scale scenario. In addition, we may need to restrict the scope of a context to a specific environment, to confer scalability, security and application performance. For example, consider the following scenario:

> A user that has a smart phone executing a *ubiquitous communicator*. This hypothetical application enables communication with a group of people, integrating functionalities of a mobile phone and of an instantaneous communicator, and has the ability to adapt the communication mechanism (e.g. text, voice, video, asynchronous messages) to restrictions imposed by the current physical environment and the available computational (and communication) resources. Hence, when the user enters an environment that establishes restrictions to communication mode at specific dates and times, the application would adapt itself to start converting voice messages to text messages, and switch itself to silent mode. For example, if the user is inside a theater hall during a movie projection, all the received calls are converted to text messages and stored in the user's text messages inbox at the device.

To adapt their communication modes in virtually any environment, applications need to deal with context semantics that may be specific to an environment. For example, a communication restriction may be associated with a specific location inside the environment and, in this case, an application may also need to deal with a particular location semantic (e.g. Room2, Hall, Cafeteria). In another environment (e.g. home, work, hospital) both the communication restrictions and the location semantics may change. Research in context-aware middleware has not yet explored these aspects of the proposed scenario.

So far, no middleware for context-aware computing properly satisfies all the requirements related to ubiquitous and dynamic context-aware environments. For example, some middleware platforms offer flexible solutions for dealing with context heterogeneity, i.e., co-existence of different context types, applications or administrative domains. In general, these solutions are not scalable in the number of clients or the volume of context information to be distributed. On the other hand, some middleware offer a scalable solution for handling and disseminating context, but restrict the types or domains of context information.

We argue that a middleware must satisfy 12 requirements to support adequately ubiquitous context-awareness scenarios. In the following sections we discuss these requirements (*R1* to *R12*) for middleware for ubiquitous context-awareness. We omitted some other requirements that have been extensively discussed in literature such as communication paradigms, mobility management and common requirements for mobile computing middleware.

R1: Distributed context management. Distributed context management promotes efficiency and scalability, since it decreases the number of clients under responsibility of a middleware infrastructure, and the network distance between a client and the management infrastructure. In order to enable efficient dissemination, context information should be delivered by the instance of the middleware's context management service that is currently closest to the client. A query to a distributed context must be decomposed in sub-queries, each one executed at the corresponding context management service, and the query results must be dynamically aggregated and delivered to the client. Distributed management also demands that clients dynamically discover context management services.

R2: Support for context evolution. A middleware for ubiquitous context awareness must support the dynamic inclusion of new context types without causing disruption in the executing application clients [5] or demanding redeployment of the system [6]. Evolution of context types may be necessary after deployment of new sensors and inference mechanisms. The challenge of this requirement is to enable applications to seamlessly continue their context-based operations, without requiring any redevelopment or redeployment.

R3: Dynamic Context Discovery. *Context discovery* is the ability of a system to detect context types associated with the new environment to which a client has moved, and identify their relationship/similarity with the context types used so far by an application. One of the responsibilities of context discovery is to support on-the-fly resolution of type-specific conflicts and ambiguities among context instances that satisfy an application's interest. For example, if an application is interested in a more precise location information for a specific device, the context-aware middleware must provide means for dynamically verifying which one satisfies the application's requirement, among several context information providers that may be currently available. This dynamic checking is required because the precision of context may vary dynamically and the information availability may change accordingly to the client's location. Moreover, some applications may be interested in context data provided by the more specific domain where a client is currently located. For example, a guide application (e.g. for a theater or museum) based on symbolic locations probably will use the location information provided within the environment where the user is being guided, even if it is less precise or accurate than other available location providers (e.g. GPS global positioning).

R4: Scope of context perception. Certain context information or privisioning mechanisms may be restricted in the domain of a physical environment or an application. For example, a research institute may use active badges to tracking the location of each employee but the usage of this information may be restricted to the institute's applications.

A developer may need to model such a restriction to context usage, which we call the *scope of context perception*. Such scope should transparently restrict the context access so that an application may automatically start accessing/perceiving the context as soon as the user enters in context scope. The ability to specify scopes of context percep-

tion decreases the number of context types and instances that a middleware has to manage and, consequently, contributes to the system's scalability. Moreover, scopes also allow for the implementation of security mechanisms for context access and distribution.

R5: Multiple mechanisms and policies for accessing context. In heterogeneous environments, where an application may be running on different devices, a middleware must offer mechanisms for adapting the policies for context access according to the available device/network resources or application requirements. For example, an adaptation may allow on-demand access to context, and the usage of optimistic or conservative policies [6].

R6: Extensible abstractions for accessing and using context. Different sorts of context information demand for specific abstractions of usage, such as specific events to which an application is supposed to react, queries with a context-specific semantics, or particular kinds of precision specification. For example, for the location context type, spatial queries such as *"which users are less than 100 meters from me"* are very particular, and do not have any equivalent for other kinds of context. MiddleWhere [15] is an example of middleware that implements a rich set of specific abstractions for location context. These abstractions specify an *interface* of context usage. They promote the correct usage of the context and its reuse for other applications with similar context-awareness requirements. Middleware systems must allow the specification of such abstractions, without limiting the evolution of the context systems (see R2).

R7: Management of application dynamic loading. In the same way as certain context information can be limited to a scope, context-aware applications also may make sense only in a certain environment. When a user moves to a new environment, he/she must be capable of discovering new applications and services as well. Middleware systems should manage the registration of environment-specific services, and allow automatic downloading of these software components when the user enters or leaves a certain domain. The main goal of this requirement is to avoid that the management of application loading monopolize user's attention by demanding his explicit input always he moves to another environment. Of course, these decisions are largely dependent on the user's interest, preferences and security policies.

R8: Abstract handling of context interest. In a ubiquitous environment, the interest of a user for a kind of context may remain unchanged, even when he continues to use the same application on another device. In this case, it is necessary to identify the real entity interested in the context, which may be either an application instance or a user. Hence, a middleware should handle context interests in a more abstract way, for example, adopting a *user-centered* approach for delivering context information.

R9: Architectural independence. In heterogeneous environments, one of the most elementary requirements for middleware systems is platform and hardware independence. In the specific case of a middleware for ubiquitous context-awareness, this translates into the support for executing applications on different architectures, as well as allowing the access to context information from different programming languages. Another requirement is the possibility to use different mechanisms for network communication (e.g. RMI, SOAP, etc.) to access context information.

R10: Decoupling between context management and inference mechanisms. A general-purpose context-aware architecture must support the modeling of complex context information and context reasoning. Most advanced modeling approaches adopt ontologies to describe context information and context reasoning, i.e., rules to infer a context from a more basic context. Ontologies provide a expressive modeling mechanism, but they require computational power to verify the model's consistency and execute inference rules. As a result, they may hardy be used in large-scale ubiquitous environments. To overcome these limitations, recently some research groups have proposed approaches of hybrid context modeling (e.g. [20]), which make use of ontologies combined with other techniques of modeling and processing, in a attempt combine the benefits of each model. However, those approaches are application-specific or context-specific.

To deal with this trade-off, we argue that mechanisms for context inference must be decoupled from context management infrastructures, i.e., should be implemented as external entities, such as an application component or context providers.

R11: Easy incremental deployment, distributed administration and standardization. Global scale context-aware solutions must enable the incremental addition of new environments and enable a decentralized environment administration. For example, a context-aware environment of an institution forms a specific administrative domain, controlled by the institution's administrator, although the environment may be a part of a global environment. An approach for ubiquitous context-awareness must also offer simple mechanisms to establish standards to be followed by most of the environments. Such standards enforce interoperability among distributed environments.

R12: Suitable programming tools for context discovery. Ubiquitous environments also demand new application development tools to ease the discovery of the available context models and to choose which context is more appropriate to implement a particular adaptation.

3. STATE OF THE ART

Research in context-aware computing has produced distributed middlewares that satisfy partially some of the requirements discussed in the previous section. According to how they promote application ubiquity, we can classify middlewares in two categories: (a) middlewares that promote *localized ubiquity*, i.e. the ubiquity is limited to a physical or application domain; and (b) middlewares that promote *global ubiquity*, i.e., support context access for a wide-scope physical **and** application domain. Sections 3.1 and 3.2 discuss how middlewares in each of these cathegories satisfy the requirements previously discussed.

3.1 Localized Ubiquity

Middlewares that promote *localized ubiquity* focus in of-

fering programming abstractions and context models that adequately deal with the ubiquity requirements in the chosen domain. In these middlewares, any concern about the performance is restricted to localized scalability, as defined in [18]. Confab [12], Gaia [16], AURA CIS [13] e PACE [10] are the most relevant representatives of these middlewares.

In general, these middlewares adopt distributed infrastructures to efficiently disseminate context information (req. R1). For example, PACE adopts a distributed publish/subscribe system. In most cases, though, either the application must previously know which infrastructure provides the context it needs (e.g. Confab) or applications must use a unique and central point of access (e.g. Gaia). In the first case, applications may become more complex, whereas in latter case, there is a central point of failure, which hinders the use of the system in highly distributed environments. Some middlewares support the concept of a dynamic environment where applications access context and share services, and support the migration between these environments. For example, in Gaia, an *active space* is a physical area (e.g. a room) in which heterogeneous network devices, such as PDAs and printers, can discover themselves, self-configure and interact with each other. However, when an application migrates to another active space, it cannot continue context-aware interactions started in the previous active space. Hence, applications experience disruptions after a migration. However, Gaia allows applications to access files created in other active spaces.

Each middleware proposes a different approach to deal with heterogeneity (req. R9). Some systems have specific implementations for portable devices (e.g. Confab), while others support user-centered computing, as Gaia. In Gaia, for example, a user can continue the same interaction in the active space even if he changes the device in use. An interesting approach is the generation of stubs and libraries for access to context information from a descriptive model of context. This approach is adopted in PACE and RCSM [21] and provides means of transparent access to the context management infrastructure, as well as independence of a programming language.

3.2 Global Ubiquity

A common approach for promoting global ubiquity is to use a *federation* of context-aware computing systems, as proposed in Nexus and CAMUS. In CAMUS [14], federations are composed of environments based on CAMUS services, which distribute context information through tuples. All the services of an environment must be registered with the Jini discovery service. An environment with a minimum set of CAMUS services for providing and using of context is called context domain. The several Jini services responsible for different domains form a federation. Thus, to access context information or use a service of a specific domain, a client must query the Jini federation, passing parameters such as the name and location of the target domain. Through this organization, applications access context by performing distributed queries. A similar approach is also adopted in SCaLaDE [1]. Nexus [9] supports federations of heterogeneous context-aware systems through a common abstract interface that each system must implemented. A context-aware system must also be registered in a service called *Area Service Register*, which assumes a similar role of DNS. As soon as the federation is built, clients can use a single query language called AWQL to access context information.

CoCo [2] and Strang *et al* [19] propose a similar federation approach, through an abstract language for the description and interoperability of context-aware systems. CoCo supports the description of workflows that specify inferences and context relationships among environments. Strang *et al* [19] propose a standard ontology to describe the facts and concepts of different domains, while suggesting the use of a middleware to implement the interoperability between two or more domains. We argue that none of the federation-based approaches is sufficient, because they are strictly limited to information dissemination and interoperability. Moreover, such approaches may cause an overload of context types when context-aware environments are integrated in global scale. Therefore, the solution hardly meets the requirements R3 and R4. In particular, federations of context-aware systems should be able to automatically discover and recognize new consumers, producers, and types of context information, as soon as they are deployed.

Hesselman *et al* [11] have proposed a bridging mechanism among heterogeneous context management systems (CMS), enabling a mapping between concepts of two different CMSs. The bridging approach allows mapping concepts such as identity, query translation, context adaptation, and context reasoning. Using this approach, an context interest described according to one CMS's interface may include context information provided by any CMS that maintains a bridge with the aforementioned CMS. Despite enabling interoperability, this approach suffers from performance and scalability limitations, since each CMS represents a central point of access and a bridge may introduce communication delays.

4. TOWARDS A MIDDLEWARE FOR UBIQUITOUS CONTEXT-AWARENESS

We propose a novel middleware approach for context-aware computing that satisfies the requirements *R1*, *R2*, *R3*, and *R4*. The main goal of the proposed middleware is to enable applications to seamlessly interact with distributed and dynamic environments for accessing context information according to statically defined expressions of context interest. The associated context management approach is based on *context domains*, each of which defines: (i) a set of context types; (ii) the kind and place for storage of context instances; (iii) the form of managing clients inside the domain; and (iv) relationship to sub-domains.

A context domain establishes a context-aware environment that applies to a specific network domain. The context types related to a context domain define a set of context concepts that are specific to a logical location, and are neither shared nor visible to context consumers outside the host domain. Thus, each context domain maintains its own context type system. However, a concept of one domain may be related to a concept of its super-domain, through a sub-typing relationship.

A *Context Management Node* (CMN) is responsible for managing a context domain and implementing the required services for storing context and disseminating it to context consumers (e.g. context-aware applications or services). Applications interact with the CMN responsible for their current domain, but this is handled transparently by a domain

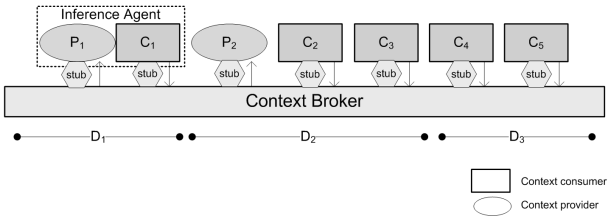

Figure 1: Component Interaction

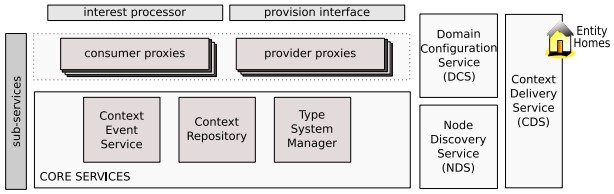

Figure 2: Architecture of a Context Management Node

discovery service. To enable efficient access to local context , each device runs a local node that is responsible for storing context published locally and for disseminating it to local applications.

Essentially, three components interact to create, disseminate and use context information: context providers (e.g., a sensor), context consumers (e.g., an application) and the Context Broker, as shown in Figure 1. The Context Broker is an abstraction of the distributed set of Context Management Nodes. Both context providers and consumers interact with the Context Broker through stubs generated by a context model processing tool, for each context type. The stub maintains all the details of how the context is retrieved, and by such enables transparent context access for providers and consumers, independently if the context information is local or not to its current domain.

Through its modularity and well-defined component interactions, the architecture manages to separate the inference mechanisms from the context model management, i.e., any context-processing agent is implemented as an external component that consumes some context required for the inference and publishes the produced/inferred context through the Context Broker. This approach avoids the use of complex context models that usually hinder efficient context dissemination.

4.1 Management of Evolvable and Distributed Context Type Systems

The relationship between context types, mentioned earlier, is the basis to implement context interoperability. For example, if a consumer describes its interest in a more general context, it may receive context instances of any of its several sub-types. Formally, a consumer interested in a context T of a domain D may receive context of types T_1 of domain D_1 and of type T_2 of domain D_2, if both T_1 and T_2 are subtypes of T. In this case, D_1 and D_2 must be sub-domains of D. In the proposed architecture, of course, context interoperability is based on the assumption that concepts in a domain are compliant with standard concepts defined in its super-domains. Moreover, the root context domain, which is the super-domain of any existing domain, must define accurately its concepts in order to allow global context interoperability (see R11).

In the underlying communication system, context information is distributed among context management nodes, and is also stored in context repositories as XML objects. Thus, stubs hide from the application any changes in the context type system, for example, as the result of the context type evolution.

4.2 Middleware Architecture

The *Context Management Node* (CMN) is the basic element responsible for managing a domain in our middleware. The current context domain of a context consumer corresponds to the domain defined at its current point of network connectivity. Thus, when a context consumer moves and changes the network segment where it is connected, the domain where it is included will also change. The CMN intermediates any context-aware interaction between a consumer and the rest of the system. In order to disseminate context to a specific consumer, the CMN uses a distributed event service based on publish/subscribe paradigm. The responsibility for delivering context to consumers is delegated to the event service of the node currently responsible for the context consumer.

A CMN is composed of the basic elements shown in Figure 2. The *Event Service* is responsible for providing asynchronous communication, delivering context information and contextual events to interested consumers. The Event Service adopts a publish/subscribe paradigm and offers a specialized API to handle subscriptions for contextual events. The *Type System Manager* maintains the dynamic context type system, solving and recognizing context types at runtime for a specific domain. The *Context Repository* maintains the database of context data. Proxies are the representatives of context consumers and context providers accessing a domain node, and implement the protocol that maintains the state of the connection to each consumer and provider. When a client migrates to another domain, the corresponding proxy migrates to the domain node, as a part of the interdomain handover protocol.

The *Context Delivery Service* (CDS) is the key service to enable efficient discovery of context information in a distributed environment. CDS maintains *Entity Homes*, i.e., a database that records any context data that describes a specific entity (e.g. a device, a person, a place). Each entity has its own home address, where the CDS obtains the domains with context instances that satisfy the consumer requirements (in terms of context type and constrains). Through the entity home, a user can dynamically inspect and control how consumers may access context for a specific entity he owns (e.g. his devices).

4.3 Implementation Issues and Testing Scenario

We have implemented the proposed architecture in Java (mixed JVM and Dalvik environments). Since our protocols are IP-based, we adopted the Service Location Protocol for discovering CMNs, whenever a mobile device becomes connected to a new network access point. The Event Service is based on the Naradabrokering[1] distributed publish/subscribe service. Our test scenario is composed of

[1] `http://www.naradabrokering.org/`

clients based on Android Platform, using the platform emulator and the Nokia N800 device. Currently, we are finishing the implementation of our testing application: a location-based application that adapts to the most specific context model and location providers, according to its current location. In this application, when the user moves to a different location, the middleware disseminates location information (either GPS coordinates or symbolic location) provided for the more specific scope (current context domain), enabling the application to show different types of maps and objects that represent places. As basic infrastructure for access to raw context data (e.g. the device's and quality of its connectivity), as well as symbolic locations, we are using our previous context provisioning middleware MoCA [17].

5. REFERENCES

[1] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli. A mobile computing middleware for location- and context-aware internet data services. *ACM Trans. Inter. Tech.*, 6(4):356–380, 2006.

[2] T. Buchholz and C. Linnhoff-Popien. Towards realizing global scalability in context-aware systems. *LNCS: Location- and Context-Awareness*, 3479:26–39, 2005.

[3] G. Chen, M. Li, and D. Kotz. Design and implementation of a large-scale context fusion network. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. The First Annual International Conference on*, pages 246–255, 22-26 Aug. 2004.

[4] H. Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland, Baltimore County, December 2004.

[5] J. Coutaz, J. L. Crowley, S. Dobson, and D. Garlan. Context is key. *Commun. ACM*, 48(3):49–53, 2005.

[6] R. C. A. da Rocha and M. Endler. Context management in heterogeneous, evolving ubiquitous environments. *IEEE Distributed Systems Online*, 7(4), April 2006. art. no. 0604-o4001.

[7] A. Dearle, G. N. C. Kirby, R. Morrison, A. McCarthy, K. Mullen, Y. Yang, R. C. H. Connor, P. Welen, and A. Wilson. Architectural support for global smart spaces. In *MDM '03: Proceedings of the 4th International Conference on Mobile Data Management*, pages 153–164, London, UK, 2003. Springer-Verlag.

[8] P. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. Irisnet: an architecture for a worldwide sensor web. *Pervasive Computing, IEEE*, 2(4):22–33, Oct.-Dec. 2003.

[9] M. Grossmann, M. Bauer, N. Honle, U.-P. Kappeler, D. Nicklas, and T. Schwarz. Efficiently managing context information for large-scale scenarios. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pages 331–340, 8-12 March 2005.

[10] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for distributed context-aware systems. *Lecture Notes in Computer Science*, 3760:846–863, 2005.

[11] C. Hesselman, H. Benz, P. Pawar, F. Liu, M. Wegdam, M. Wibbles, T. Broens, and J. Brok. Bridging context management systems for different types of pervasive computing environments. In *First International Conference on Mobile Wireless Middleware, Operating Systems and Applications (MOBILWARE)*, Innsbruck, Austria, February 2008. ACM Press.

[12] J. I. Hong. The context fabric: an infrastructure for context-aware computing. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 554–555, New York, NY, USA, 2002. ACM Press.

[13] G. Judd and P. Steenkiste. Providing contextual information to pervasive computing applications. In *Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on*, pages 133–142, 23-26 March 2003.

[14] S. L. Kiani, M. Riaz, S. Lee, and Y.-K. Lee. Context awareness in large scale ubiquitous environments with a service oriented distributed middleware approach. In *ICIS '05: Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05)*, pages 513–518, Washington, DC, USA, 2005. IEEE Computer Society.

[15] 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.

[16] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *Pervasive Computing, IEEE*, 1(4):74–83, Oct.-Dec. 2002.

[17] V. Sacramento, M. Endler, H. K. Rubinsztejn, 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.

[18] M. Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10–17, Aug. 2001.

[19] T. Strang and C. Linnhoff-Popien. Service interoperability on context level in ubiquitous computing environments. In *Proceedings of International Conference an Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet*, L'Aquila, Italy., 2003.

[20] M. Strimpakou, I. Roussaki, C. Pils, N. Kalatzis, and M. Anagnostou. Hybrid context modeling: A location-based scheme using ontologies. In *3rd International Workshop on Advanced Context Modelling, Reasoning and Management*, Pisa, Italy, March 2006.

[21] 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.