

Domain-based Context Management for Dynamic and Evolutionary 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
{rcarochoa, endler}@inf.puc-rio.br

ABSTRACT

Research in context-aware computing has produced a number of application prototypes, frameworks, middlewares and models for describing context. However, development of ubiquitous context-aware applications is still a difficult task because current middleware systems are focused on isolated and static context-aware environments. Currently, applications require a global knowledge of the context-aware infrastructures in order to establish context-based interactions, and they suffer of problems such as disruptions when a context-aware environment evolves. The goal of this thesis is to develop a distributed middleware for context-aware computing that allows applications to maintain context-based continuous interactions, even in a highly dynamic environment. In order to achieve this goal, we propose a context management strategy based on context domains. We argue that this approach supports four important requirements for context-aware ubiquitous applications: distributed management of context, support for seamless evolution of context-aware systems, dynamic context discovery and domains of context perception.

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*

Keywords

Context-aware computing, middleware, ubiquitous computing, context management

*This work is being funded by CNPq Research Grants nos. 552.068/02-2 (ESSMA Project) and 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.

MDS'07 November 26-November 30, 2007, Newport Beach, CA, USA
Copyright 2007 ACM 978-1-59593-933-3/07/11 ...\$5.00.

1. INTRODUCTION

Research in context-aware computing has produced a number of application prototypes, frameworks, middlewares and complex models for describing context. However, most middlewares are restricted to very specific applications, centralized architectures, limited physical domain or scope. In a truly ubiquitous computing scenario, a user moves through environments in different domains, which may maintain their own sensors and mechanisms for inferring context.

In order to enable such broader scenario, we envisage several context-aware systems (CAS), providing different types of context information, maintaining different context models and covering different physical or logical places.

Most middlewares for context-aware computing are special purpose and restricted to specific application and physical domains such as smart meeting rooms [16], smart tourist guides and web content adaptation [3]. Currently, context-aware environments are isolated and independent, hindering communication among each other. We call these isolated environments context-aware islands, because they hinder implementation of applications with cross-environment interest in context information, i.e., when the interest of application is a combination of contextual situations provided by context providers in different environments.

Some middlewares deal with these limitations by offering distributed platforms for context management [10, 15, 9] or federations of context-aware systems [6, 14, 2]. However, we argue that such middlewares do not provide efficient solutions because they have either limited generality or scalability. On one hand, more generic and flexible solutions are usually not scalable, since they add an excessive complexity to the context management mechanism. On the other hand, scalable solutions often support only limited abstractions and paradigms for context usage.

In a ubiquitous scenario, the adoption of current middleware approaches for context-aware computing has several drawbacks. Firstly, applications need a global knowledge of all CAS in order to identify which one provides the context information they are interested in. Moreover, if more than one CAS contains the desired information, clients must solve dynamic conflicts and inconsistencies among different contexts in order to choose which context information is more appropriate for its current purposes, what increases the complexity of applications. Finally, in dynamic and evolvable environments, changes in context models may cause disruption in client interactions if they were statically prepared to

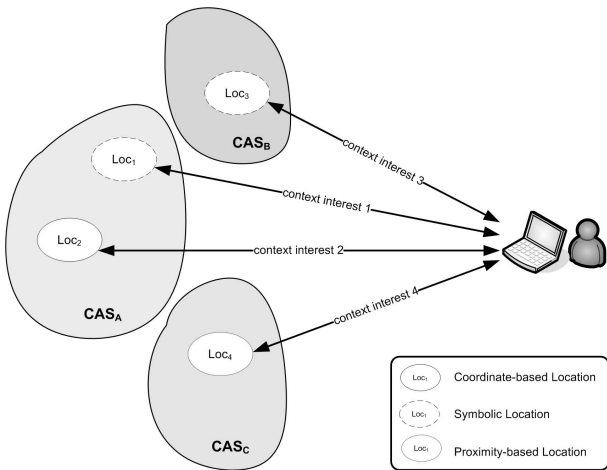


Figure 1: Client interaction with multiple context-aware systems

interact to previously known distributed CAS and handle their context information.

This thesis investigates mechanisms for context modeling and management that enable the composition of such dynamic and evolvable context-aware environments in order to provide ubiquity to context-aware applications. The contribution of the thesis is the development of a distributed middleware architecture based on the concept of *context domains* that enable sharing and interoperability among CASs in an evolvable and dynamic environment.

2. PROBLEM DEFINITION

A context-aware system (CAS) is a computational environment that comprises context sources (e.g. sensors), context information types and instances, and the context management infrastructure responsible for them. An application interested in context information provided by a specific CAS, maintains a *context interest*, which may be described either as a synchronous or an asynchronous access of the context.

Figure 1 shows a scenario where a client application maintains a context interest distributed among three different CASs, which provide specific types of location information for a certain entity (e.g. a device or a person). In the example, since the independent CASs do not specify an interrelationship among their location types, the application should be responsible for solving possible inconsistencies among the location obtained through each context interest and to dynamically evaluate which location satisfies its requirements.

The proposed scenario demands middleware for enabling context-aware applications to dynamically switch among dynamic, distributed and evolvable CASs, maintaining seamlessly their context interests, despite of application mobility¹.

We argue that a middleware for such a scenario must fulfill four requirements: distributed context management, support for seamless evolution of context-aware systems, dynamic context discovery, and domains of context perception.

Distributed Context Management.

¹In this work, we consider application mobility as a result of device mobility.

In a macro-scale ubiquitous scenario, context-aware architectures must be distributed in order to allow efficient and scalable dissemination of context. However, a distributed architecture may introduce new problems for context-aware clients, as they may require a prior knowledge of the distributed middleware infrastructure responsible for disseminating a specific context they are interested in. Thus, distributed context management must be implemented in conjunction with services for dynamic discovery of context services and for transparent distribution of context information.

Support for Seamless Evolution of Context-Aware Environments.

Context-aware environments are inherently dynamic as a result of frequent replacement and addition of new types of sensors, applications or context inference mechanisms. These changes may require updates of the context models, of the context databases and of the means that the middleware executes context queries. The challenge of this scenario is to accommodate such changes in the environment without compromising active context interests. If an evolution interferes in the result of a context interest then applications must perceive such change, without requiring to rebuild their context interest. On the other hand, the creation or change of context types must not compromise the consistency of global context type systems.

Dynamic Context Discovery.

A distributed context-aware infrastructure may offer context information from different types and sources that describe a same context an application is interested in. The selection of which one is appropriate for an application purpose may depend on context meta-attributes, such as precision and accuracy, and may dynamically change accordingly to the availability of new context information. For example, when a device enters a physical environment with its own location mechanism, applications interested in this device's location should become aware of the availability of this new type of location information and evaluate if this new type of location is appropriate for their purposes. Ideally, the middleware, and not the application, should be responsible for choosing the most adequate context information among a dynamic set of available ones. We call this functionality *dynamic context discovery*.

Domains of Context Perception.

The usage of certain context information may be restricted to some domains, environments or applications. In this case, by restricting the access and *perception* of the context, we may increase the scalability of the middleware. Moreover, it reduces the number of CAS the may be involved in the conflict resolution of multiple context interests.

3. STATE-OF-THE-ART

Classical work in context-aware computing has developed centralized and application-specific solutions, where context-discovery is limited to sensor discovery, such as in Context Toolkit [17]. These middlewares and frameworks offer solutions for restricted and small-size environments, with *localized* scalability. More recent middlewares offer access context information in distributed repositories or context

management infrastructures. Confab [11], Gaia [16], AURA CIS [12] and PACE [9] are the most representative middleware systems that support context distribution.

Confab [10] maintains context information in distributed tuple-spaces called *infospaces*. Each infospace is a repository responsible for storing one or more context types. An application interested in a certain context, builds a context query using the address of the responsible infospace. Infospace servers maintain groups of infospaces, and they may run in the same device of the application, if it manages local information types. In order to handle queries over distributed infospaces, Confab offers a query processing service [8], which distributes queries over distributed infospaces and composes the query results. Although distributed infospaces contribute to decrease the context management overhead in a distributed environment, this distribution is not kept transparent to applications, which must know what infospace contains the desired context information. AURA CIS [12] implements a similar approach, though offering transparency among distributed context repositories.

Gaia [16] adopts the concept of *active spaces*, which are physical spaces where devices in a heterogeneous network, such as PDAs and printers, can discover each other, auto-configure and dynamically start a context-aware interaction. If an application moves to another infospace, it loses its context interests. These solutions take into account that management of a context type and storing context instances are complementary tasks. However, this assumption makes the management of context difficult in distributed environments where some concepts and types could be shared, as well to interpret a context interest when a client moves to another environment. PACE [9] is another distributed middleware, which focuses in offering a flexible context model and advanced context-based programming abstractions. PACE is organized in layers that provide, besides context management, an interface for distributed context queries and a layer for adaptation, which maintains a repository of reusable adaptation abstractions. Applications are not statically linked to distributed repositories and they use a catalog and meta-attributes to discover which repository satisfies their context requirements. In a more recent work, Springer *et al* [18] explored distributed context management to integrate highly heterogeneous environments based on 4G communication technologies. This work, however, does not support evolutionary environments, as described in our scenarios.

Another important issue in research in middleware for context-aware computing is the support for flexible context models that enable the description of environment complexity and context inference methods. In particular, there is a crescent interest in using ontologies for modeling context and ontology-based reasoning to implement inference of high-level context information.

However, in distributed CAS, ontology-based context models may hinder an efficient and scalable access to context information [20]. In fact, there is a trade-off in the adoption of complex context models and the distributed and efficient access to context information. This problem has motivated the development of hybrid context models, which use ontologies in conjunction with other mechanisms for modeling and processing context, in order to obtain the best of each modeling approach.

Architectures with distributed CAS demands for context modeling approaches that offer sharing of context among

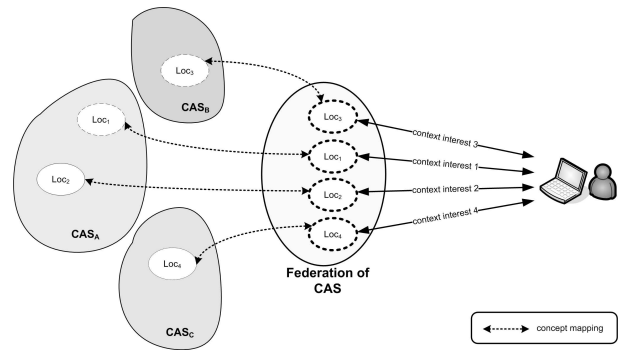


Figure 2: Federation approach for integration among context-aware distributed systems

different CAS, inter-domain dynamic exchange of context and the support for scoping context. To the best of our knowledge, there is no modeling approach that efficiently supports these three requirements in a distributed architecture.

Another approach for allowing distributed CAS is based on federating context-aware services, as shown in Figure 2 and implemented in middlewares such as Nexus and CAMUS. In these approaches, there is a common interface that allows an independent CAS to share its context models with another CAS.

In the middleware CAMUS [13], a CAS federation is composed by environments based on CAMUS services, which disseminate context information as tuples, in order to increase dissemination efficiency. Each service of an environment must be registered in a Jini discovery service. A CAMUS context domain is an environment that support a minimum set of CAMUS services. The set of Jini services responsible for each CAMUS domain composes a federation. In order to access context information or to use a service of a specific domain, client must query the Jini federation, using parameters such as the name and localization of the domain. Through this approach, client may access and query for distributed context information. SCLaDE [1] is another middleware that implement a similar approach. In Nexus [7], a federation contains heterogeneous CAS. In order to allow inter-CAS interoperability, each CAS must implement an abstract interface and register itself in the *Area Service Register*. A client may access context information provided by a federation, using a query language called AWQL.

We argue that approaches based on federation of CAS are not enough to satisfy the requirements of the proposed research problem, because they restrict the research challenge to an interoperability issue and to information dissemination. In particular, federation approaches should allow establishing interrelationships among context of different CAS, in order to support context interoperability, i.e. deliver a context from a different type or provider if the context describe the same environmental state and satisfies the client requirements of precision and accuracy. Additionally, federation should implement context scoping to decrease the overhead of context management for CAS and the amount of context types and instances that are involved in the selection of a client interest.

CoCo [2] and Strang *et al* [19] have proposed a different approach for interoperability among distributed CAS. They

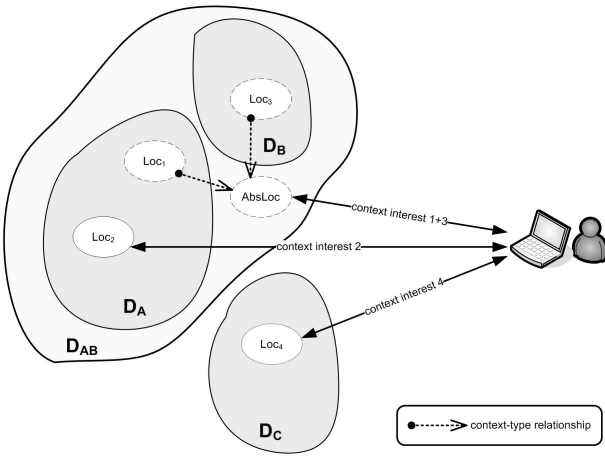


Figure 3: Domain-based approach for context management

propose an abstract language and ontology that a CAS must use to publish their context in a common infrastructure. In this case, clients have no more a notion of CAS distribution but they access context on a centralized repository. Moreover, the authors agree that describing a general purpose language for context interoperability is a challenge.

4. DOMAIN-BASED CONTEXT MANAGEMENT

This work proposes distributed context management based on *context domains* to support distributed context-aware environments. A context domain establishes: (i) a set of context types; (ii) responsibility for the storage of context instances; (iii) responsibility for managing clients inside the domain; and (iv) sub-domains.

Figure 3 shows the reorganization of the motivating scenario through context domains. When two domains share some common characteristics, such as a same interpretation of context types or they are in a same administrative domain, they are considered parts of a same super-domain. In the figure, D_A and D_B are sub-domain of D_{AB} domain. A context domain substitutes an isolated context-aware system.

We define *context consumer*, an application (or inference agent) that registered an interest for a context. As an application, a context consumer runs in a device. Thus, if the device migrates to a domain then we say that all the respective context consumers migrate as well.

The context types defined on a context domain establish a set of context concepts that are specific to a certain logical location and, thus, they are not shared or perceived by context consumers outside the domain. Each context domain maintains its own context type system. However, a concept of a domain may be related with a concept of its super-domain, through a sub-typing relationship.

This context relationship is the basis to implement context interoperability. For example, if a consumer describes interest in a more general context, it may receive context instances of any of its several sub-types. In the example shown in the figure, if a consumer is interested in context *AbsLoc* (an abstract location), it may also receive notifica-

tions about changes in the context Loc_1 and Loc_3 . We argue that this approach decreases the number of context consistency checks to be managed by consumers. For example, in Figure 3, the consumer has to manage three context interests whereas in the previous examples there were four interests to manage. A context interest is described in terms of any of the following parameters: type (which may be more or less specific), entity characterized by the context, domain of the context type and context meta-attributes, such as precision. The middleware infrastructure is responsible for dynamically selecting the context information that best fits a current application interest.

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 . Clearly, in the proposed architecture, context interoperability is based on the assumption that concepts in a domain are compliant to standard concepts defined in its super-domains. Moreover, the root context domain, which is the super-domain of each existing domain, must define accurately their concepts in order to allow global context interoperability.

In our context modeling approach, a context type comprises attributes, context queries and contextual event. Additionally, attributes can have meta-attributes, such as precision and accuracy. A more complete description of our context model can be obtained in [5].

4.1 Middleware Architecture

The *Context Management Node* (CMN) is the infrastructure responsible for managing a domain. When a context consumer enters a new domain, it discovers the domain node through the *node discovery service*, which behaves similarly to a Jini discovery service. The current context domain of a context consumer corresponds to the domain defined on 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 may change. The CMN intermediates any context-aware interaction of a consumer with the rest of the system. In order to disseminate a 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 responsible for the context consumer.

In order to allow transferring responsibilities for a context consumer when it migrates between domains, the middleware implements an inter-domain hand-off procedure. The node discovery service is responsible for detecting domain changes and to trigger the inter-domain hand-off protocol, which is transparent to context consumers.

A context consumer can interact with three types of context domains: local, current and remote. Remote domains are common domains supported by distributed context-aware middleware. In Figures 1 and 2, every interaction with a CAS is remote, i.e., a non-local interaction. A local domain is limited to the device where applications are running. A local domain establishes context types that are used only for applications running in the device, and maintains context information published by locally placed context providers (e.g. a GPS sensor). Moreover, context-aware applications may be interested in using their own context types, which should not be globally published or maintained. In order to

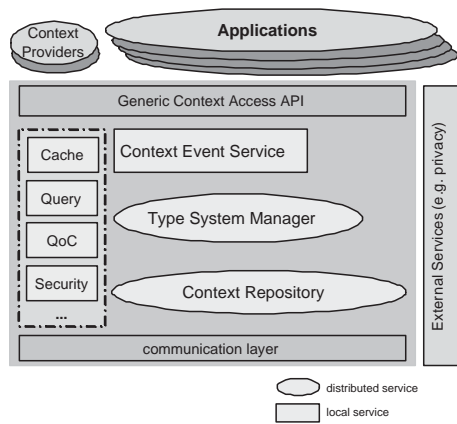


Figure 4: Middleware Services

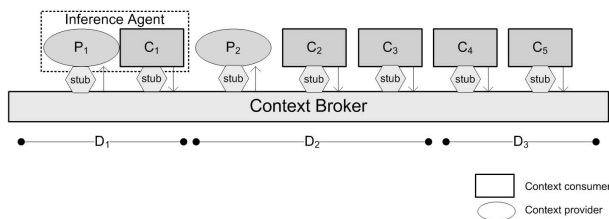


Figure 5: Component Interaction

maintain local context domain, a light-weight instance of the middleware services runs in each (client) device. By maintaining local context domains, our approach allows context-aware applications to access efficiently local context, even when there is no network connectivity, as well to provide services as context caching that contribute for the overall performance of the system.

A context management node is composed of the basic elements shown in Figure 4. 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 dynamical context type system, solving and recognizing context types at runtime for a specific domain. The *Repository* maintains the database of context data.

4.2 Component Interaction

In our approach, essentially three components interact to create, disseminate and use context information: context providers, context consumers and the Context Broker, as shown in Figure 5. The *context provider* is an entity responsible for publishing a certain context information; it can be the generator of a raw sensor data or an inference agent, which infers a new context from another context. The context consumer is an entity interested in a given context information. The Context Broker is an abstraction for distributed context management nodes.

As a result of this component interaction, the architecture maintains inference mechanisms outside the context model management, i.e. any inference agent must be implemented as an external component that consumes the context re-

quired to produce inference and publishes through Context Broker the new inferred context. This approach avoids the usage of complex context models that hinder efficient context dissemination. In order to implement inference agents based on ontologies, we developed an approach for exchanging concepts between an ontology-based model and our modeling approach [4].

4.3 Deployment of Context Types

As shown in Figure 5, both context providers and consumers interact with the Context Broker through context type *stubs*. They encapsulate the underlying code required to request or publish a specific instance of a context type. The developer of a context-aware application includes the stubs of the required context types, which map a context type to object-oriented language constructions. From the perspective of an application developer, the access to context information is strongly typed.

Stubs are generated during the deployment of a context type, which involves two main steps: context modeling and the context model processing. The first step consists of modeling the new context information using an XML-based approach. In an XML file, the context modeler² specifies attributes, characteristics, relationships with previously specified context, quality-of-context attributes and queries for synchronous context request.

In the context model processing step, a *Context Tool* reads the XML file and executes the following tasks: (i) Validates the XML syntax and the context model; (ii) Updates the context type system and initializes the repository for storing the new context information; (iii) Generates a library containing the language bindings for the access operations of the deployed context. So far, we have just implemented the Java language binding for context access. Dependencies among context types are also included in the binding file.

In the underlying communication system, context information is distributed among context management nodes as XML objects. Context repositories also maintain context information as XML objects. Thus, stubs hide changes in the context type system, resulting from a context type evolution, allowing the application to access the previous snapshot of the context type. The system fails in maintaining the validity of context interests (implemented through stubs) when the user that implements a change in the context model removes an information of a type (such as an attribute or an event) or includes an incompatible change. In both cases, the Context Tool alerts the user about the consequences of the changes in deployment.

5. CONCLUSIONS

We have developed an initial prototype to validate the proposed approach. The repository of a context management node stores context information and event as XML objects and disseminate asynchronously using a distributed event service. We are developing a light-weight middleware instance to run in resource limited devices (PDAs based on Windows Mobile). We are planning to test some policies for caching context in such devices to increase the efficiency of context access, especially when the network connectivity is intermittent.

²The user that models a new context.

As a part of our main current investigations, we are validating our context modeling approach. In particular, we are investigating how the assumption of decoupling context models and context provision management (Section 4.2) may limit our usage scenarios.

We are still choosing a test scenario for our architecture, which will be based on the usage of following context types: location, local device resources, device profile and environmental context obtained from Crossbow wireless sensors. In order to test the scalability of our middleware architecture, we are planning to simulate a scenario of multiple context aware applications in a university campus composed of multiple context domains.

6. 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, 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.
- [3] 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.
- [4] R. C. A. da Rocha, M. A. Casanova, and M. Endler. Promoting efficiency and separation of concerns through a hybrid model based on ontologies for context-aware computing. In *Pervasive Computing and Communications Workshops, 2007. Proceedings of the Fifth IEEE Annual Conference on*, 19–23 March 2007.
- [5] R. C. A. da Rocha and M. Endler. Supporting context-aware applications: Scenarios, models and architecture. In *Proc. of the XXIII Simpósio Brasileiro de Redes de Computadores (SBRC)*, volume I, Curitiba, Brazil, May 2006.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for distributed context-aware systems. *Lecture Notes in Computer Science*, 3760:846–863, 2005.
- [10] 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.
- [11] J. I. Hong and J. A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 177–189, New York, NY, USA, 2004. ACM Press.
- [12] 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.
- [13] 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.
- [14] S. L. Kiani, M. Riaz, Y. Zhung, S. Lee, and Y.-K. Lee. A distributed middleware solution for context awareness in ubiquitous systems. *rtcsa*, 0:451–454, 2005. 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05).
- [15] O. Riva. Contory: A middleware for the provisioning of context information on smart phones. In *ACM/IFIP/USENIX 7th International Middleware Conference (Middleware'06)*, Melbourne (Australia), November 2006.
- [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] 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.
- [18] T. Springer, K. Kadner, F. Steuer, and M. Yin. Middleware support for context-awareness in 4g environments. In *WOWMOM '06: Proceedings of the 2006 International Symposium on World of Wireless, Mobile and Multimedia Networks*, pages 203–211, Washington, DC, USA, 2006. IEEE Computer Society.
- [19] T. Strang and C. Linnhoff-Popien. Service interoperability on context level in ubiquitous computing environments. In *Proceedings of International Conference on 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.