# A SURVEY OF CONTEXT-AWARE MIDDLEWARE

Kristian Ellebæk Kjær
Department of Computer Science
University of Aarhus
Åbogade 34
8200 Århus N, Denmark
email: argo@daimi.au.dk

## Abstract

This paper provides a survey of a chosen set of context-aware middleware systems, and categorises their properties and use according to a taxonomy. An overview of each system is provided, as well as descriptions of the different properties.

## Keywords

Middleware, Context-awareness.

## 1 Introduction

This paper presents a survey of context-aware middleware systems. Based on the the surveyed systems, a taxonomy for categorising properties of such systems has been created. The taxonomy is based on the capabilities of the systems, based on what their individual authors find important.

This paper provides a short introduction to middleware in section 1.1 and then introduce the taxonomy in section 2. The systems are described in section 3.

### 1.1 Middleware

We assume a definition of middleware as software systems which provide an abstraction layer between an operating system and applications running in a distributed environment.

The role of middleware is to provide an additional layer of abstraction suitable for a specific type of applications. The intended type of applications might vary form any type of "distributed system", or as narrow as "agents in Java". However, middleware is normally intended for a specific type of distributed system, and even though a given middleware systems claims to be suitable for general distributed systems, underlying assumptions often implies certain limitations for the usefulness of the system.

In traditional distributed systems, the goal of the middleware has been to *hide* heterogeneity and distribution by providing ways of treating remote resources as if they were local. In wired, static environment, this has proven useful, but in dynamic, wireless environments it breaks down, since applications often need to base decisions on information about distribution and the environment. Instead, middleware systems for Pervasive Computing focus on providing suitable abstractions for dealing with heterogeneity and distribution *without* hiding them, and in some cases even provide information about distribution and heterogeneity as context information.

### 1.2 Types of Context

To understand context-aware middleware, it is useful to understand different types of context. The most widely used context information today is arguably *location* and *proximity*. These are examples of context that is external to the computer systems which use them, but internal context, such as available disk space is useful, but often not considered context, and thus not modelled as part of the model of context. Other forms of context include information collected by *sensors*, which can range from biometric information to measuring the amount sunlight at a location.

## 2 Categorising Middleware

The categorisation is based on surveying existing context-aware middleware. The surveyed systems are very different and some are pure middleware while others are more complete infrastructural systems offering services for managing entire physical environments. The systems also vary in what constitutes *applications* build with the system.

The systems are categorised according to the taxonomy depicted in figure 1. In the following sections, we detail each of the major categories of the taxonomy.

**Environment** A middleware system makes explicit or implicit assumptions about the environment it is to be used in. Some middleware assume the existence of an infrastructure which offers services needed by the middleware and applications. We say that these systems live in an *infrastructure* environment. Other systems only assume that devices have some method of communication and does not rely on external services. We say that systems build with this kind of middleware is *self-contained*.

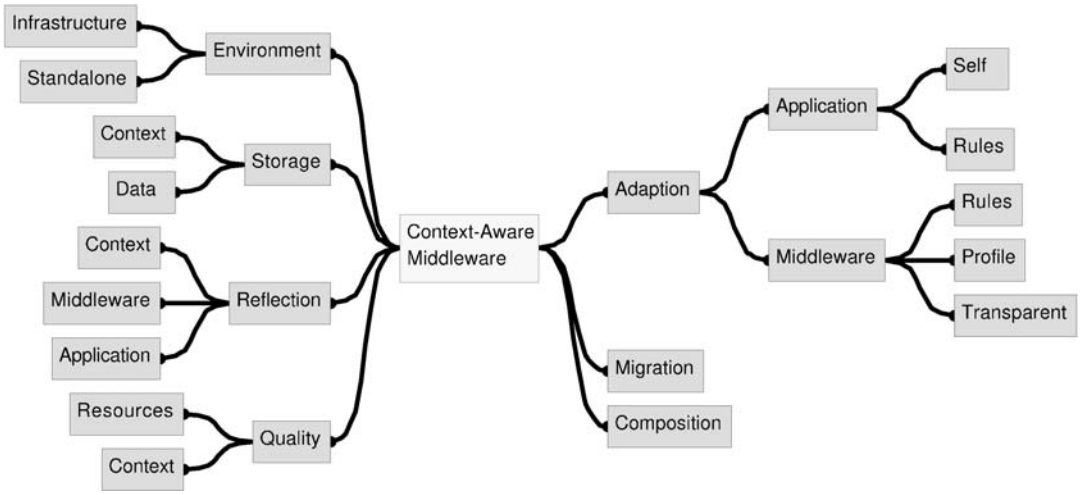


Figure 1. Taxonomy of Context-Aware Middleware showing major categories, sub categories, and sub-sub categories

**Storage** Some systems provide a context-aware data store which order data based on context information, allowing it to be retrieved based on certain context-parameters. For example, some systems provide file systems where data is ordered according to the current context. Other systems provide centralised storage facilities for context information, allowing applications to retrieve it.

**Reflection** Reflective mechanisms are mostly known in programming languages, but some middleware systems offer reflection of different parts of the systems. In such systems, the reflective mechanisms are typically available through *meta-data* the applications can access. The meta-data can represent either

- the application itself,
- the middleware,
- or context information.

Closely tied with meta-data is the concepts of *reification* and *absorption*. Reification is the process of representing entities as meta-data and absorption is the process of altering the entities by altering their representation. That is, that the entity described in the meta-data adapts to changes in the meta-data.

**Quality** Quality is a measure of how well a service can be performed or how good data is. In the case of context-aware middleware quality is mostly concerned with quality of service, how well a resource can be provided. However, some systems provide quality measures of the offered context.

**Composition** Some middleware do component composition based on contextual events. For example, entities might be composed with all entities in their vicinity, or composition might be changed if some context event occurs.

**Migration** Some systems provide migration of entities. Some of the systems merely provides mechanisms for migrating running code when the application decides, possibly based on context, while other systems migrate entities automatically based on context. This is closely tied with *adaption* (section 2, where different parts of the systems adapts to context.

**Adaption** When context-information is available, systems can *adapt* to changes in the context. Different parts might all use context in different ways, but most middleware systems do not use context-information on all parts of the system. Adaption to changes might happen in *middleware* or in the *applications*. If adaption takes place at the middleware level, there are three sub-categories:

- *Transparent.* The middleware reacts to changes in context without the application being aware of it.
- *Profile.* The middleware receives a profile from the application detailing what kind of service is is interested in. It is then the responsibility of the middleware to adapt so that it can provide a service as close to the requested as possible.
- *Rules.* Rules are typically of the form if *a* then *b*. Rules are provided by either applications or by users, and indicate what action the middleware should or must take when *a* happens.

When adaption is the responsibility of the application, the programmer is free to use context in any way imaginable. However, some middleware systems provide methods

for invoking certain actions in the application based on context changes in the form of rules, typically with a call-back to the application.

**Other Aspects** Besides the above, other aspects of context-aware middleware are important for understanding the different systems. These include:

- *Basic entity.* The basic entity in a system is the entity which constitutes the application build on top of the middleware. Some middleware systems are intended for general purpose applications, but others instead group existing applications or assume specific programming paradigms.
- *The underlying OS.* In this context, OS should be understood very widely. All middleware systems run on top of some sort operating systems, or support different operating systems. It might also be the case that some parts of the system runs on e.g. PDAs while the infrastructure is based on services provided by web applications, as in the case of Cooltown [16].

Other interesting aspects include fault tolerance, disconnected operation and context history.

## 3 Middleware Systems

Context-Aware Middleware systems have very different characteristics. To better understand systems, we have applied the taxonomy introduced in section 2 to a number of middleware systems to show its usefulness.

Table 1 provides an overview of the different systems, and shows which parts of the taxonomy each of them support. The rest of this section describes each of these systems.

### 3.1 Aura

Aura [11, 13, 15, 20] is a *task* oriented system for infrastructural environments. It runs on top of an ordinary desktop operating system, and explores the notion of *personal aura*, a system which supports the user in performing tasks. Services for management of tasks, applications, and context are provided. Unlike most other context-aware middleware, Aura does not support building applications, but instead relies on existing applications to act as *service providers* for services like text editing or playing sounds.

Tasks are controlled by a *Task Manager*, which handles migration of tasks, while services are provided by an *Environment Manager*. Tasks are abstract representations of a collection of services comprising the task. When a user moves from one environment to another, the representation of the task is moved, and service providers for the task are instantiated at the new location.

Aura provides a *Context Observer* to manage context. Context information is used to derive the *intent* of the user. The context observer merely collects context, and reports changes to the Task- and Environment Managers. Depending on the detail of the collected context, the Context Observer might be able to derive the current task, location, and intent of the user. The current task is used for proactively loading the current task, while location is used to migrate tasks e.g. from the home to the office of the user. The intent is used for both tasks and location. If the user is working at home, but have a meeting scheduled at 10am and leaves the home computer, the Context Observer might derive that the user is leaving for the office, and migrate the current task without intervention from the user. If the Context Observer is unable to derive the location or intention of the user, e.g. because of insufficient sensors in the environment, the user must explicitly indicate this to Aura.

### 3.2 CARMEN

CARMEN [3] is intended for handling resources in wireless settings assuming temporary disconnects. It uses *proxies*, mobile agents residing in the same CARMEN environment as the user. If a user moves to another environment the proxy will migrate using wired connections. Each mobile user has a single proxy, which provides access to resources needed by the user. When migrating, the proxy makes sure that resources are also available in the new environment. This can happen by: *moving the resources with the agent*, *copying the resources*, *using remote references*, or *re-binding to new resources which provide similar services*. The method is determined by inspecting the *profile* of the device.

Each entity in CARMEN is described by a *profile*. *User profiles* contains information about preferences, security settings, subscribed services etc. *Device profiles* define the hardware and software of devices. *Service component profiles* define the interface of services and *Site profiles* group together the profiles which all belong to a single location. Thus, context information in CARMEN describes the entities which make up the system.

### 3.3 CARISMA

CARISMA [4, 5, 6]deals with adaption of middleware depending on the needs of the applications.

Profiles for each application are kept as meta-data of the middleware and consists of *passive* and *active* parts. The passive parts define actions the middleware should take when specific context events occurs, such as shutting down if battery is low. The active information defines relations between services used by the application and the policies that should be applied to deliver those services. The active part is thus only used when the application requests a service.

Different environmental conditions may be specified, which determine how a service should be delivered. At any time, the application can use reflection to alter the profile kept by the middleware through an XML representation.

Table 1. Categorisation of Context-Aware Middleware Systems.

| Middleware | Environment | | Storage | | Reflection | | | Quality | | Composition | Migration | Adaption | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | Middleware | | | Application | |
| | Infrastructure | Self-contained | Context | Data | Context | Middleware | Application | Resources | Context | | | Transparent | Profile | Rules | Rules | Self |
| Aura | √ | | √ | √ | | | | √ | | √ | √ | √ | | | | |
| CARMEN | √ | | √ | | | | | | | √ | √ | | √ | | | |
| CARISMA | | √ | | | | √ | | √ | | | | | √ | | | |
| Cooltown | √ | | √ | | | | | | | √ | | | | | | √ |
| CORTEX | | √ | √ | | | | √ | √ | | | | | | √ | | |
| Gaia | √ | | √ | √ | | | | √ | | √ | √ | | | | | √ |
| MiddleWhere | √ | | √ | | | | | | √ | | | | | | | √ |
| MobiPADS | | √ | | | √ | √ | √ | | | √ | √ | | √ | | | √ |
| SOCAM | √ | | √ | | | | | | | | | | | | √ | |

To deal with conflicts between profiles, CARISMA adopts a micro-economic approach [6], where a computing system is modelled as an economy where consumers makes a collective choice over a limited set of goods. In this case, the goods are the *policies* used to provide services, not the resources providing them. The middleware plays auctioneer in a action protocol, where each application submits a single, sealed bit on each alternative profile. The auctioneer then selects the alternative which maximises the sum of bids. To determine the bid each of the applications are willing to pay, functions which translate from profile requirements to values are defined. Like profiles, these functions may be changed at any time through reflection. This type of protocol makes sense because CARISMA delivers the *same* service to all participants.

### 3.4 Cooltown

The Cooltown project [2, 8] is intended to support wireless, mobile devices to interact with a web-enabled environment. The basic principle is that devices, people, and things have a web-presence identified by a URL, which provides a "rich" interface to the entity. Users interact with the web-enabled environment using PDAs to interact with the available web-services. As such, Cooltown expects wireless Internet access when users interact with the system. URLs are passed between devices in local device to device interaction. E.g. a projector might receive a presentation by receiving a URL to the file.

Context in the system is closely tied to the physical environment. For example, an infrared beacon at the entrance of a room will emit a URL which points to the page of the room [2]. When a PDA loads this page, the PDA acts as an interface to the room, thus changing behaviour based on location context. Other types of context might be used by web-applications by providing web-applications with other context like time or activity. The main principle in the collection of context is that it is provided by web-clients. Depending on which sensors the clients haves, web interfaces can adapt to the context they provide. Types of context about the physical world includes [8]: *where*, *when*, *who*, *what*, and *how*.

The context is integrated with a model of the physical world, consisting of places, people and things, and relationships between them. Relationships include: *Contains*, *isContainedIn*, *isNextTo, and isCarriedBy*, and the list is extensible. Relationships are directional, so like hyperlinks they can be navigated in one direction, making them suitable for presenting as web pages. Relationships have properties, and can be subtypes of other relationships. The state of the model is updated automatically by sensing mechanisms ranging from infrared beacons to GPS.

The main modules in the architecture are: *Web Presence Manager*, *Description*, *Directory*, *Discovery modules*, *Autobiographer*, *Observer*, and *Control*.

Besides these modules, Cooltown offer tools to build web-presence services (applications).

### 3.5 CORTEX

The CORTEX project [9, 22] project is concerned with research other than context-aware middleware, but have proposed a middleware to deal with

> "Autonomous mobile physical objects that cooperate with other objects, either mobile or static, by capturing information in real-time

from sensors event messages propagated in a MANET" [22].

The middleware is based on sentient objects. A sentient object senses and views the behaviour of neighbouring objects, reason about it, and manipulates physical objects accordingly. Sentient objects dynamically discovers each other, and share context information.

To support sentient objects, CORTEX provides a middleware based on component frameworks, each of which provides a service to the sentient objects: *Publish-Subscribe*, *Group Communication*, *Context*, and *QoS management*.

Publish-Subscribe is used for discovery. While the other component frameworks support communication, context retrieval and inference, and arbitration of resource allocation.

The resulting middleware is configured at deployment time and can be reconfigured at run-time through a reflective API to adapt to changes in the environment.

### 3.6 Gaia

The Gaia Operating Systems [19, 18]is intended to be a *meta-operating system*. That is, a distributed middleware system providing functionality similar to an operating system. Gaia builds on the notion of an *active space*, coordinating heterogeneous devices in a physical space, typically a single room. Like operating systems, it provides: *program execution*, *I/O operations*, *file-system access*, *communications*, *error detection*, and *resource allocation*.

Program execution is supported by the *component manager core*, which allows any application to upload components to any node of execution in the active space. I/O operations are supported by device drivers on each node, and the functionality is exported to the rest of the active space using distributed objects. Gaia utilises the *Context File-System*, which stores files based on representation of the context. Both synchronous and asynchronous communication is supported through RPC and events. Applications can register for event notification in case of errors, and react accordingly. Finally, Gaia manages resources throughout the active space.

Gaia is structured like traditional filesystems with a kernel providing the necessary services and applications build with a application framework on top.

Gaia differentiates between location, context, and events and although they can all be seen as different kinds of context, they are handled by different components. Context is collected by *context providers* and higher level context, such as activity, can be inferred from low level context. An additional presence service deals with which entities are present in an active space. Four basic types of entities are supported: application, service, device, and person.

Context is represented by first-class predicates and more complex context is represented by first-order logic operations, such as *and* and *or*. Applications are notified of changes in context through events, and can react accordingly.

#### 3.6.1 Context File System

The Context File System [18] builds a virtual hierarchy based on context-information tags on files and presents a directory structure based on context predicates. For example, files associated with the context **location=room-031 && present=Esben** can be retrieved by entering the directory **/location:/room-031/present:/Esben**.

The Context File Systems is build with a single server governing the namespace, while actual files are imported into the active space using existing distributed file systems, such as NFS.

### 3.7 MiddleWhere

MiddleWhere [17] provides advanced location information to applications and incorporates a wide range of location sensing techniques in a model for location. Location information originates in *Location Providers* and is stored in a spatial database. A reasoning engine uses the location information from different providers to determine location and a location service uses the spatial database and the reasoning engine to provide location with a certain probability.

The location model is hierarchical and deals with three different kinds of location: *points*, *lines*, and *polygons*. Each is represented by coordinates and a symbolic name. Location is represented as GLOBs (Gaia LOcation Byte-string). For example, a desk could be represented as **Building1/3/338/Desk1** or as **Building1/3/338/(2,4,0)**, meaning that Desk1 in room 338, floor 3 of Building 1 is located at coordinates (2,4,0) with respect to the coordinate system of the room. The room will have coordinates with respect to the floor, the floor with respect to the building, and the building with respect to global coordinates. In this case, the desk is represented by a point. Polygons are used for representing rooms, hallways or spaces within rooms, while lines can be used for representing doors between two rooms.

#### 3.7.1 Quality of Location Information

The systems deals with quality of the location information. The quality is measured according to *resolution*,*confidence*, and *freshness*. Resolution differs widely between different location sensing techniques. For example, a person using a card-reader to enter a room will tell the system that the person is somewhere inside the room while GPS has a resolution down to perhaps 10 meters. An RF badge might have a resolution of 1 meter. Confidence is a measure of how precise the sensor is in terms of probability that the object is within the sensed area. This probability originates in the sensors which register the object and in the case of multiple sensors, the information is fused to yield a single value. Freshness is based on the time since the last

sensor reading, and each type of sensor has an associated temporal degradation function which, based on freshness, degrades the confidence in the information.

### 3.8 MobiPADS

MobiPADS [7] is a middleware system for mobile environments. The principle entity is *Mobilets*, which are entities that provide a service, and which can be migrated between different MobiPADS environments. Each mobilet consists of a slave and a master. The slave resided on a server, while the master resides on a mobile device. Each pair cooperate to provide a specific service. Services are composed by chaining them together in specific order, and the slave mobilets on the server are nested in the same order. This provides reconfiguration based on different requirements.

MobiPADS is concerned with internal context of the mobile devices, which is used to adapt to changes in the computational environment. Thus, context types include: *processing power*, *memory*, *storage*, *network devices*, *battery etc*. Each of these have several subtypes, e.g. *size* and *free_space* for memory and storage. Mobilets are provided with changes through context events, which they subscribe to. Higher order context is derived by an *Environment Monitor*, which subscribe to event sources and has the same characteristics as other event sources.

Adaption takes place in either the middleware based on system profiles, or by letting mobilets adapt to the events they receive. Based on the requirements in the profile, the service chains can be reconfigured to deal with e.g. a constrained environment, based on programmer provided alternatives service chains. Applications have access to reflective interfaces for context, service configuration, and adaption strategies, and can change them to obtain a different service from the middleware.

### 3.9 SOCAM

SOCAM [12] is based on the idea of using ontologies to model context. The model is then used by an interpreter to reason about context. The SOCAM architecture consists of: *Context Providers*, *Context Interpreters*, *a Context Database*, *a Service Location Service*, and *Context-aware Mobile Services*. *Context Providers* provide external or internal context, which can be used by the mobile services directly or by *Context Provider* to provide higher-order context. Externally, the Context Interpreter acts as a Context Provider. Context is represented as instances of the ontology model.

The Context Interpreter consists of a Context Reasoner, and a *Context Database*, which contains instances of the current ontology, either from users or Context Providers. The context is updated by a triggering mechanism with different intervals. Context Providers register with the *Service Location Service*, thus allowing Mobile Services to locate them. The *Mobile Services* can obtain context either by querying the located Context Providers, or by registering for specific events. SOCAM supports *rules* for specifying which methods should be invoked on events. The rules are predefined and loaded into the Context Reasoner.

#### 3.9.1 Representation of Context

SOCAM represents context as a formal ontology as predicates in OWL. The middleware supports reasoning about context, so that high level context can be derived from observed context by the Context Interpreter. The ontologies are either a generalised ontology, or a domain specific ontology which is "bounded" with the generalised ontology, or "re-bounded" if the context changes. The domain specific ontology may, for example, be re-bounded if the context shifts from an office location to a car. It is the responsibility of the Service Locating Service to load new context ontologies when applications ask for location context.

## 4 Related Work

Most middleware system have an, implicit or explicit, notion of types of context. Some explicitly categorises context, and uses the categorisation for dealing with different kinds of context. Others have an implicit notion of context types, illustrated by letting different parts of the system deal with different kinds of context, sometimes without even considering it context, e.g. Roman et al [18].

A recent survey of context-aware systems by Baldauf and Dustdar [1] introduce a layered architecture for understanding context-aware systems, but does not focus specifically on middleware, or present existing systems in very much detail.

Gaddah and Kunz [10] provide a survey of different middleware paradigms targeted at Mobile Computing . However, they do not describe context-aware systems in very much detail, and describe context-awareness as a separate paradigm from e.g. reflective and tuple-space based middleware systems, even though some context-aware middleware systems use these paradigms to provide context information and integration. As described in this paper, it is possible to use these mechanisms for actually implementing context-awareness. Especially reflection has been used for supporting adaption to changes in context.

Strang and Popien [21] have surveyed modelling of context, describing key-value, graphical, object oriented, logic based, and ontology based models, but they do not describe middleware systems.

Henricksen et al [14] provide an overview of the state-of-the art of context-aware systems. They survey five different systems, and compare them in terms of *requirements* of context-aware systems. However, they restrict themselves to looking at *general* systems which spans multiple layers of a proposed layered architecture, consisting of application components at the top layer, decision support tools, context repositories, context processing components,

and context sensors and actuators at the bottom layer. A general middleware system is defined as a system which supports general context, and not e.g. just location data. In contrast, we also survey properties of less general systems.

## 5 Concluding Remarks

This paper has provided a survey of context-aware middleware systems and a taxonomy for categorising them. The taxonomy is founded in the capabilities of the surveyed systems.

### 5.1 Future Work

It is of interest to survey additional context-aware middleware systems to fit them into the presented taxonomy, or even determine whether they fit the taxonomy. It can be imagined that some additions or even changes might prove useful, although we believe the existing taxonomy will fit most systems. Systems which have not yet been fitted into the taxonomy includes: Capnet, EasyLiving, MiLaN, one.world, Oxygen, and countless others.

Additional work on understanding types of context might also be useful, especially if coupled with research in models of context to determine what types of context can be represented in a given model.

### 5.2 Conclusion

The current state-of-the-art of context-aware middleware explores quite different approaches to support pervasive and mobile computing based on context information. All of the middleware systems provide some method of adapting to changes in the context, and methods for collecting context, but otherwise use different entities and have different focus. We believe that no single middleware system is appropriate for all settings, so an understanding of what existing systems do is useful for choosing how to create or select a middleware system for a specific need, and this survey provides a starting point for understanding the possibilities.

## References


[1] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing, forthcoming*, 2004.

[2] J. Barton and T. Kindberg. The Cooltown user experience. Technical report, Hewlett Packard, February 2001.

[3] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli. Context-aware middleware for resource management in the wireless internet. *IEEE Transactions on Software Engineering*, 29(12):1086–1099, 2003.

[4] L. Capra. Mobile computing middleware for context-aware applications. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 723–724, New York, NY, USA, 2002. ACM Press.

[5] L. Capra, W. Emmerich, and C. Mascolo. Reflective middleware solutions for context-aware applications. *Lecture Notes in Computer Science*, 2192:126–133, 2001.

[6] L. Capra, W. Emmerich, and C. Mascolo. Carisma: context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29(10):929 – 45, 2003/10/.

[7] A. Chan and S.-N. Chuang. Mobipads: a reflective middleware for context-aware mobile computing. *IEEE Transactions on Software Engineering*, 29(12):1072 – 85, 2003/12/.

[8] P. Debaty, P. Goddi, and A. Vorbau. Integrating the physical world with the web to enable context-enhanced services. Technical report, Hewlett-Packard, September 2003.

[9] H. A. Duran-Limon, G. S. Blair, A. Friday, P. Grace, G. Samartzisdis, T. Sivahraran, and M. WU. Context-aware middleware for pervasive and ad hoc environments, 2000.

[10] A. Gaddah and T. Kunz. A survey of middleware paradigms for mobile computing. Technical Report SCE-03-16, Carleton University Systems an d Computing Engeneering, July 2003.

[11] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. *IEEE Pervasive computing*, 1(2):22–31, April–June 2002.

[12] T. Gu, H. K. Pung, and D. Q. Zhang. A middleware for building context-aware mobile services. In *Proceedings of IEEE Vehicular Technology Conference*, May 2004.

[13] U. Hengartner and P. Steenkiste. Protecting access to people location information. In D. Hutter, G. Müller, W. Stephan, and M. Ullmann, editors, *SPC*, volume 2802 of *Lecture Notes in Computer Science*, pages 25–38. Springer, 2003.

[14] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for distributed context-aware systems. In R. Meersman, Z. Tari, M.-S. Hacid, J. Mylopoulos, B. Pernici, Ö. Babaoglu, H.-A. Jacobsen, J. P. Loyall, M. Kifer, and S. Spaccapietra, editors, *OTM Conferences (1)*, volume 3760 of *Lecture Notes in Computer Science*, pages 846–863. Springer, 2005.

[15] G. Judd and P. Steenkiste. Providing contextual information to pervasive computing applications. In *PERCOM '03: Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, page 133, Washington, DC, USA, 2003. IEEE Computer Society.

[16] T. Kindberg and J. Barton. A web-based nomadic computing system. Technical report, Hewlett-Packard, August 2000.

[17] A. Ranganathan, J. Al-Muhtadi, S. Chetan, R. H. Campbell, and M. D. Mickunas. Middlewhere: A middleware for location awareness in ubiquitous computing applications. In H.-A. Jacobsen, editor, *Middleware*, volume 3231 of *Lecture Notes in Computer Science*, pages 397–416. Springer, 2004.

[18] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, and K. Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74 – 83, 2002/10/.

[19] M. Román, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, pages 74–83, Oct–Dec 2002.

[20] J. P. Sousa and D. Garlan. Aura: An architectural framework for user mobility in ubiquitous computing environments. In *WICSA 3: Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture*, pages 29–43, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V.

[21] T. Strang and C. L. Popien. A context modeling survey, September 2004.

[22] C.-F. Sørensen, M. Wu, T. Sivaharan, G. S. Blair, P. Okanda, A. Friday, and H. Duran-Limon. A context-aware middleware for applications in mobile ad hoc environments. In *MPAC '04: Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 107–110, New York, NY, USA, 2004. ACM Press.