Supporting Generic Context-Aware Applications for Mobile Devices

Ralph Löwe, Peter Mandl
CC Information Systems and Management
Munich University of Applied Sciences
Munich, Germany
e-mail: rloewe@hm.edu, mandl@cs.hm.edu

Michael Weber
Institute of Media Informatics
University of Ulm
Ulm, Germany
e-mail: michael.weber@uni-ulm.de

Abstract—**Most of the advances in the field of context-aware middleware center on the research of the components necessary to create one. Although this is vital and gives important insights, it does not immediately enable us to build context-aware applications more easily. Too few projects reach the state of a usable software product. In this paper we describe an approach to separate the building of context-aware applications from the middleware. The centerpiece is an interface to independently define a context-aware application and its state. Additionally we provide a coarse protocol definition for the communication between the instances of a context-aware middleware by the example of a client/server system for mobile devices.**

Context; context-awareness; context-aware middleware; mobile computing; context-aware application

I. Introduction

At the beginning of the 90s Mark Weiser [1] defined the term ubiquitous computing with a vision of highly interconnected devices. Like glasses disappear in the background and enable a person to see, the computer should become an integral and invisible part of everyday life [2].

He described a new age of computing which follows the age of personal computing. Computers should help to better complete the everyday tasks without being directly visible to the user. This could transform the computer to something normal instead of something magical [2].

To achieve this goal he proposed challenges for the hardware and software of ubiquitous systems. One example are three classes of devices: small tabs, medium-sized pads and large boards. Another example are the three different types of network adapters and protocols [3, p. 77f].

On top of this hardware he motivated the creation of a ubiquitous network which connects all devices and allows them to interact with each other. He proposed a "micro-kernel" operating system as one solution [1]. Based on top of this stack he described the applications as *"the whole point of ubiquitous computing."* [3, p. 80].

If we compare this vision with the current state we find smartphones, pads, digital television and smartboards, which all include many different classes of devices. Every device contains various sensors and communication adapters, which additionally could be used as sensors as well [4].

In the software layer the vision and the state of the art differ more. Since the underlying systems must be connected using software, there are many requirements that such a system must fulfill. For example the ability to automatically interconnect devices, exchange context information about the user or migrate running processes.

This paper describes an approach of a service as part of a context-aware middleware. In detail we will show the interface for the communication between the components in a mobile environment. It enables the interchangeability of the providing and using software components and therefore provides generic context-awareness.

II. Problem

The central challenge is the interaction between the user and the ubiquitous system. It is still complex and demands a grand part of the user's attention. The field of context-aware computing tries to transfer the concept of context from human-to-human into human-to-computer interaction [5]. This improves the communication from the perspective of the user.

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [5, p. 305].

From this definition the primary usage of context can be derived. It can be utilized to separate relevant from irrelevant items or sort in accordance of relevancy. But it also opens new challenges for the software. Context needs to be disseminated from the source to the application that uses it. The collected context must be interpreted. Beyond that the application needs to adapt itself based on the context of the user [6].

An additional challenge is the simplification of the development of ubiquitous applications [7]. It is important to bridge the gap between hardware, sensors, operating system, network and the application. This could simplify the development process from the perspective of the developer.

A possible solution to both problems is the use of a context-aware middleware. In the research of context-aware middleware the first part of this term is explained very well and in depth. *"A system is context-aware if it uses context to* *provide relevant information and/or services to the user, where relevancy depends on the user's task."* [5, p. 30].

Mostly the term middleware is left out [8]. One of the reasons might be that the word middleware is widely used. But it influences the meaning of the whole and should not be neglected.

Middleware is *"a software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying [...]"* system. *"In addition to solving the problems of heterogeneity, middleware provides a uniform computational model for use by the programmers of [...] distributed applications"* [9, p. 33].

Middleware offers independent interfaces for applications and if an application is built using these interfaces, it should be portable to another instance of the same middleware type. The implementation behind the interface can and does vary and sometimes provides vendor-dependent solutions. However, the decision to use those is comprehensible for the developer. One example of middleware is the common object request broker architecture (CORBA) [10].

In the field of context-aware computing efforts have been made to create such middleware in order to simplify the development of context-aware applications. It is indispensable to develop a software in order to research its functionality. Additional important insight can be gained if we do not only research the functional capability but also the appropriate appliance.

This includes the research of standards [11, p. 57]. The central advantage of middleware is possible because of interfaces and its established standards for middleware applications. Hence, one solution for context-aware middleware would be to define standards and particularly interfaces to create independent context-aware applications. This would result in a better comparability as well as the possibility to benchmark context-aware middleware.

III. Related Work

There has been much research to create services for context-aware middleware. A milestone in this sector has been the Context-Toolkit [12] and the Java Context Awareness Framework (JCAF) [13]. Periodic overviews of the current context-aware middleware have been created [14–17]. These works offer a basis for the generalization of context-aware middleware.

Some effort has been made to make context-aware applications replaceable or domain-independent [18–21]. The framework ACORD-CS [22] describes a UML-based model to define a context-aware application. The system offers four different components and is able to generate a context-aware application. Context monitors listen to changes of context and adaptation can be performed solely based on context policies [18]. This approach takes the viewpoint of the context-aware application developer.

Kalaiselvi et al. describe a generic context-aware middleware for smart homes, that offers a graphical user interface to configure simple rule-based applications [23].

Yau et al. offer a CORBA-based approach which describes a context-sensitive object request broker. Additionally, they extended the interface definition language into a context-aware interface definition language for the purpose of building context-aware applications [20].

Other than developing software, testing software offers important insights, too. Ye et al. describe a mathematical model to compare two context-aware middleware products [19]. This supports the availability and possibility to create a generic model for a context-aware applications.

The analysis of related work revealed that most approaches focus on the implementation of the middleware and the distribution of context neglecting to also focus on the context-aware application itself together with the middleware aspects.

IV. Application Model

In our opinion a context-aware middleware must not only handle context, but also the state of the application itself. Only if it knows which adaptation is possible and in which state the application is, it can decide if an adaptation based on a change in context must be performed.

The process that a context-aware middleware must support, can be subdivided into five tasks [6, p. 307ff].

- **Acquisition** is the sensing and subsequent gathering of context. The source of context attributes could be based on sensors, combinations of context attributes and external data.
- **Representation** of context is important for the exchange of context attributes. Through transformation the type can be changed [24].
- The **storage** of context mostly requires hard drive capacity and enables a historical collection and therefore analysis of context.
- **Interpretation** of context in respect to the used application model lets a context-aware middleware decide which change should be performed. Much research has been made to test interpretation algorithms on domain problems.
- Finally the context-aware application can perform **adaptation** and therefore change its state to perfectly fit the requirements.

A model of generic context-aware applications must support all of these tasks and enable them to work correctly. A context-aware middleware should be able to support the developer of the context-aware application at every step.

The sensing of context attributes can be very domain-independent. Thus, once the acquisition of a single context attribute has been implemented, it should easily be usable by another application.

The representation of context is the first challenge. At the sensor the context attributes start as simple values. In combination with the identifying sensor a context attribute is basically a key-value pair. Therefore, we assume that a context model can be reduced to key-value pairs and later be transformed to the appropriate model [24] required by the interpretation.

Afterwards the interpretation must retrieve the context including the current values and perhaps the history of these. Additionally, domain-specific information about the interpretation method must be present to perform the interpretation algorithms.

Finally the adaptation of the context-aware application can be performed. For this functionality to work we need hooks inside the application to interact with or defined services that should be invoked.

Orthogonal to the process which a context-aware middleware must support, we can find different kinds of context-aware applications. Early work in the field of context-awareness differentiated between four categories of applications [25]: contextual information, contextual commands, automatic contextual reconfiguration and context-triggered actions.

This categorization groups context-aware applications in the broader sense. In the narrow sense it can also be applied to single features. Since the structure and processes in the same category are very similar to each other, we identified patterns and common tasks for each category.

Using this knowledge, we developed a service for a context-aware middleware and published the architectural approach in a former paper [26]. While developing it was necessary to create an interface between the client and the server. It separates the complex process of context representation, storage and interpretation from the context-aware application and facilitates the interaction on the level of context sensing and adaptation.

Furthermore, the interface represents the application model. Its main purpose is the communication of context-aware applications and it additionally enables them to independently process themselves. The state of a context-aware application can be communicated and, for example, the result of an interpretation can be returned and applied. Any context-aware middleware client could implement this interface and communicate with any middleware server, which serves this interface.

For the presentation of the design of the interface we use the simplified entity relationship model based on Barker's notation. It is used because of the increase in readability and efficient use of drawing space.

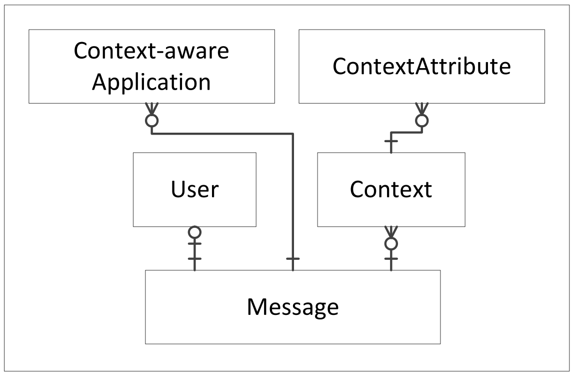


Figure 1: message level (simplified).

Since the interface is based on communication, it starts at the message level. Figure 1 shows that a message can contain many context bulks, which themselves consist of many single context attributes. This reflects the key-value context model on the message level and allows the collection of context from different measured times or users to be bundled in one message. Using a key-value context model on the message level does not restrict the usage of other models in the server component. But it requires a antecedent transformation to the target context model.

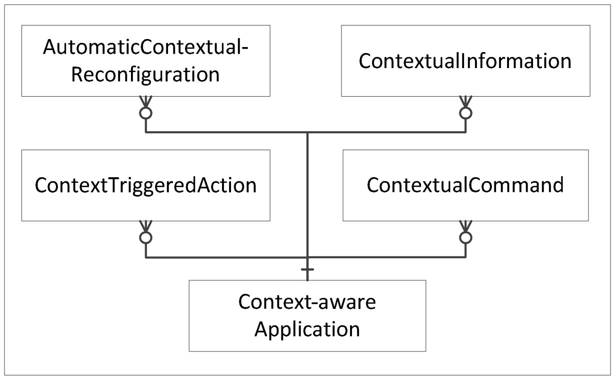

AutomaticContextual-Reconfiguration
ContextualInformation
ContextTriggeredAction
ContextualCommand
Context-aware Application

Figure 2: context-aware application.

Additionally to the identity part as a context attribute, the user is exclusively modeled as a single entity. This enables the separation of authentication and context-awareness.

The context-aware application which is shown in figure 2 contains the context-aware features and their current states. At this level we defined a context-aware application to be consisting of zero or many items of the known groups of context-aware software which stand for features. We decided to use these four categories, because the patterns of structure, adaptation and communication are very similar in each group.

The interpretation of context is an issue which can be handled centrally for all categories. A criterion describes a method for the interpretation of a current context, e.g. to calculate the relevance of multiple options. Using this definition, the interpretation can be decoupled and implemented generically. Moreover multiple criteria could be combined and can be assigned a weight to describe the relevance for the interpretation. This enables the hybrid combination of interpretation algorithms [26].

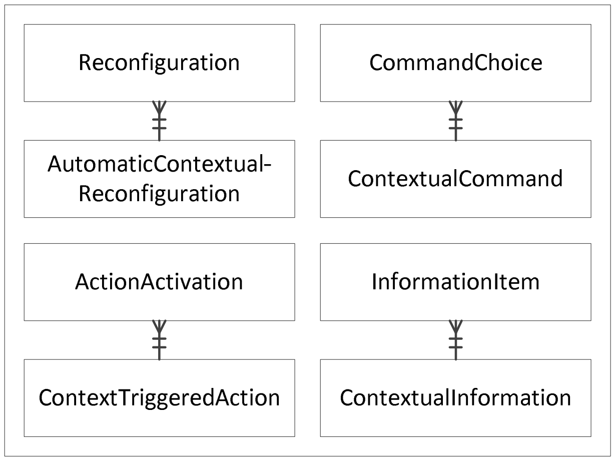


Figure 3: feature groups (simplified).

Although the attributes of each entity are different the basic structures shown in figure 3 always represent a list of options. This is necessary for the interpretation to work correctly, because the criteria require multiple options. Furthermore the items each differ in their attributes.

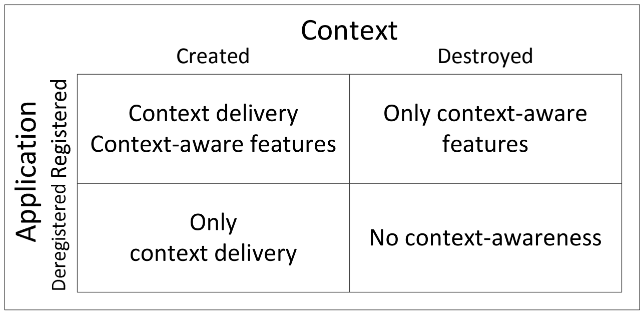
We called the items of a **contextual information** feature information items. They each need an attribute for the content of the information to be transferred. Additionally the rating of the interpretation process must be propagated, because it can be relevant to the presentation. The contextual information object holds the criterion and offers configuration through a property-based structure. Herein the information source, for example the URL of an RSS-Feed, can be specified. The context of the items can be derived from the information source and each item is assigned its context. As a special case, contextual information is the only feature where the items are filled in a phase before the interpretation. This is necessary due to the content-based nature of this feature.

Automatic contextual reconfiguration is able to use a set of single reconfigurations. Since the configuration can be very application specific, it only offers a way to explicitly identify one reconfiguration. Additionally, a pre-selected reconfiguration can be specified. On the upside this enables the middleware to know if the configuration should be changed and adapt only in this case. On the downside this could lead to network traffic if the configuration is changed manually. But since an update of context is likely to occur more often than a manual reconfiguration, this seems appropriate. Each reconfiguration can be supplemented by defined context attributes. This creates the connection between the raw configuration and the context attributes which are necessary for the context interpretation.

The structure of a **contextual command** is very similar to the automatic contextual reconfiguration. Only the names of the items differ. Instead of reconfigurations the command offers a list of choices. The interpretation is very similar to the automatic contextual reconfiguration. But we decided to separate them, because the handling inside the client is very different. While the reconfiguration can actively change components of the context-aware application when it is running, the contextual command only changes the future execution flow of the application.

The other context-aware features are only relevant for a running application. Since **context-triggered actions** can be executed even if the context-aware application is not running, they need to be registered persistently. Furthermore it is common to have more independent conditions which activate an action. Evaluating these conditions results only in two choices: to execute or not to execute the action. We designed this rule-like logic as a context-triggered action, which has many action activations. This is similar to a rule, which has an action as root and multiple or-conditions as branches.

Using the message structure, we defined types, which are represented as an attribute on the message level. We defined the usual user-handling like login and logout including the error states related to this progress. Delivery types for the creation, actualization and destruction of context are supported, too. Additionally we allow the registration, actualization and deregistration of a context-aware application. Using the states of context and application, we support four different modes of context-awareness, which are shown in figure 4.



Application \ Context	Created	Destroyed
Registered	Context delivery Context-aware features	Only context-aware features
Deregistered	Only context delivery	No context-awareness

Figure 4: modes of context-awareness (simplified).

They are based on one connection between one mobile client and one server. One server could support multiple clients, which each could have different modes of context-awareness. So if a context is created and updated regularly and the relevant context-aware application is registered, we have the highest grade of context-awareness. All available context attributes are delivered to the server and can be interpreted. Additionally, all context-aware features are active. If the context is destroyed, then the application can still be active, since the context could be delivered via other clients which are registered as a context source.

If the context is destroyed and no application is registered, only the sole connection is kept alive. When a user is logged out, then the connection is terminated. The term **logged in** does not mean that the user is aware, that his device is connected. This can be automatically performed by a middleware client in the background of the context-aware application. For example in the case of context-triggered actions no attention is directly needed for the application respective the user to be executed.

Using this set of actions we can deliver context attributes, register context-aware applications and their context-aware features. A server can now independently perform an interpretation and calculate the new application state. The next step is the effective adaptation in the client and is communicated using asynchronous communication [9, p. 163]. Since the manual features like contextual information and automatic contextual reconfiguration require synchronous communication, we additionally implemented the request-reply pattern [9, p. 203f] for these two.

In addition to the four context-aware features it is possible to listen to normal context changes and react to these events directly. We provide a simple listener pattern for these cases which acts as a low level interface.

Since the communication is solely handled at the message level, additional communication patterns can be integrated as well. Because context attributes are transferred as key-value pairs, additional attributes can be added. Since single attributes are separated from each other, it enables the developer to focus on each attribute individually. The model of the context-aware application can be extended as well but should be verified strictly.

Mobile devices specific tasks e.g. a connection loss in the case of no signal reception of the mobile or minimizing data volumes and power usage are handled centrally in the client as well. The interface has been optimized for the usage patterns of mobile devices as well. For example only necessary data is communicated and single packets can be piggy-backed.

V. Proof Of Concept

The whole context-aware middleware was developed using Java. We chose this programming language because it is very common and understandable. For the storage of persistent values the object-relational mapping framework hibernate [27] was used. A whole functioning reference implementation including server and a client for Google Android has been published as an open source project [26].

To implement the interface we choose the Google Protocol Buffer framework. Since it uses a binary representation of the data, the packets are smaller in comparison to an XML-based interface. This provides an optimal bandwidth usage and enhances the speed of the interaction especially for the communication with mobile devices.

Furthermore Google Protocol Buffers are implemented using an open specification and libraries for commonly-used languages are fully implemented. This enables any programming language used on any platform and embedded system to use or implement the stack, if it is not yet implemented.

```
message ContextualInformation {
 required int32 id = 1;
 required string label = 2;
 required string type = 3;
 repeated Criterion criteria = 4;
 repeated Configuration
   configuration = 5;
 repeated ContextualInformationItem
   items = 6; }
message ContextAwareApplication {
 required int32 id = 1;
 required string label = 2;
 repeated ContextualCommand commands = 3;
 repeated ContextTriggeredAction
   actions = 4;
 repeated ContextualInformation
   informations = 5;
 repeated AutomaticContextualReconfiguration
   reconfigurations = 6; }
```

Listing 1: Definition using Google Protocol Buffer.

The definition of contextual information and context-aware application is shown in listing 1. Both are modeled as messages, which contain multiple attributes. Attributes, which are marked **repeated**, can be repeated zero or more times. The numbers behind the equals sign are the unique numbered tags, which identify the fields of the message [28].

Besides the client the project includes a complete test facade, which uses the interface via TCP/IP. Test cases for all context-aware features, context delivery and multiple clients have been implemented. It can be used to check the functionality of the underlying interface.

The server uses a object oriented context model and transforms the attributes from the key-value model. For each application feature a single proxy is created, which represents the application state.

As conceptual proof we mapped the known parts of the project Cyberguide [29] to our context-aware application model. The main purpose of this context-aware system is a navigational, communicational and informational service for a tourist. The challenge here lies in the separation of generic and application-specific features.

The cartographer service, which offers a simple view of where the user is positioned can be implemented as context listener and therefore does not need to be registered. The librarian service can be registered as a collection of contextual information features. These could react based on activity and location. The information about the buildings could be sorted using a **nearest** and an **activity** criterion.

Cyberguide also offers a messenger service which can be registered as another set of context-aware features. This enables the tourist to send and receive information for example to the owner of the exhibit. For the interaction components automatic contextual reconfigurations can be created. Herein another contextual information feature can be registered and could sort the persons referenced to the nearest exhibit, the availability of the person and their current activity.

In detail the generic interface was concretized via single feature components. For example we implemented a combo box, which automatically selects items based on the context of the user. Another example includes a map component, which displays information on nearby zones. For the display of news a contextual information using RSS feeds has been implemented. The enriching and bundling of single feed items is handled in the server as a plug-in module.

As another example we implemented the an indoor location system like Active Badge Location System [30] and we could use some generic parts which were created implementing the tourist guide.

Therefore it is possible to replace application-specific parts of the mentioned context-aware systems using generic ones. This allows the reuse of these components in future context-aware applications and helps the developer to focus on the domain. The implementation of complex context models, interpretation algorithms and storage schemes can be handled separately from the context-aware application.

Once-implemented components, which are solely based on the interface, can be reused for other context-aware applications. We showed an example of one component in a formerly published paper [26]. Using this methodology future applications are simpler to build and once existing applications are migrated to this system, they are easier to maintain.

If the context-aware middleware supports the interface, all needed context attributes and criteria, then a context-aware application can be transferred from one context-aware middleware to another without changing it. But the experiences concerning middleware show us that this is a rare case.

VI. Conclusions and Future Work

The final conclusion of this paper is, that the standardization of context-aware middleware is necessary. This should enable us to better compare the approaches on the middleware and application level and could systemize the research in context-aware middleware more efficiently.

We showed that the usage of an open specification and the definition of an interface can qualitatively enhance the development of context-aware applications. The described interface is a proposal as opposed to a fully defined and comprehensive standardization. Further work and cooperation is needed to reach a comprehensive but simple and stable specification. Independently from our solution we showed the necessity and possibility to create and compare such standards.

This could enable further research such as software benchmarking or functional test suites for context-aware middleware. The alternative could be that the industry creates its own standards, which are comparably incompatible like the researched context-aware middleware. There are many examples in the past where such incompatibilities forestall future innovations.

Another insight was that context-triggered actions are not found very often in context-aware applications. This could be due to the lack of awareness of a running action if a specific context condition is met. On the contrary this offers more invisibility of the context-aware system than the other three features and should in our opinion be used more often.

References

[1] M. Weiser, “The computer for the 21st Century,” IEEE Pervasive Computing, vol. 99, no. 1, pp. 19–25, Jan. 1991.

[2] M. Weiser, “The world is not a desktop,” ACM Interactions, vol. 1, no. 1, pp. 7–8, 1994.

[3] M. Weiser, “Ubiquitous computing,” IEEE Computer, Hot topics, vol. 26, no. 10, pp. 71–72, 1993.

[4] P. Bahl and V. N. Padmanabhan, “RADAR: an in-building RF-based user location and tracking system,” in Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064), vol. 2, 2000, pp. 775–784.

[5] A. K. Dey and G. D. Abowd, “Towards a better understanding of context and context-awareness,” in CHI 2000 workshop on the what, who, where, when, and how of context-awareness, 2000, pp. 304–307.

[6] N. Malik and U. Mahmud, “Future challenges in context-aware computing,” proceedings of the IADIS, pp. 306–310, 2007.

[7] G. D. Abowd, “What next , Ubicomp ? Celebrating an intellectual disappearing act,” in UbiComp, 2012.

[8] M. Satyanarayanan, “Pervasive computing: vision and challenges,” IEEE Personal Communications, vol. 8, no. 4, pp. 10–17, 2001.

[9] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, Distributed Systems. Pearson Education, 2012, p. 1063.

[10] F. Bolton, Pure Corba. Sams Publishing, 2002.

[11] D. A. Norman, The invisible computer. Cambridge, Massachusetts, USA: MIT PRESS, 1998.

[12] D. Salber, A. K. Dey, and G. D. Abowd, “The Context Toolkit: Aiding the Development of Context-Enabled Applications,” in Proceedings of the SIGCHI conference on Human factors in computing systems the CHI is the limit - CHI ’99, 1999, pp. 434–441.

[13] J. E. Bardram, “The Java Context Awareness Framework (JCAF) – A Service Infrastructure and Programming Framework for Context-Aware Applications,” in PERVASIVE 2005, 2005, pp. 98–115.

[14] M. Baldauf, S. Dustdar, and F. Rosenberg, “A survey on context-aware systems,” International Journal of Ad Hoc and Ubiquitous Computing, vol. 2, no. 4, pp. 263–277, 2007.

[15] G. M. Kapitsaki, G. N. Prezerakos, N. D. Tselikas, and I. S. Venieris, “Context-aware service engineering: A survey,” Journal of Systems and Software, vol. 82, no. 8, pp. 1285–1297, 2009.

[16] D. Romero, “Context-Aware Middleware : An overview,” Revista Electrónica Paradigma en Construcción de Software, vol. 3, pp. 1–11, 2008.

[17] P. Bellavista, A. Corradi, and M. Fanelli, “A Survey of Context Data Distribution for Mobile Ubiquitous Systems,” ACM Computing Surveys, 2013.

[18] R. P. Pinto, E. Cardozo, P. R. S. L. Coelho, and E. G. Guimarães, “A domain-independent middleware framework for context-aware applications,” Proceedings of the 6th international workshop on Adaptive and reflective middleware held at the ACM/IFIP/USENIX International Middleware Conference - ARM ’07, pp. 1–6, 2007.

[19] C. Ye, S. C. Cheung, J. Wei, H. Zhong, and T. Huang, “A study on the replaceability of context-aware middleware,” in Proceedings of the First Asia-Pacific Symposium on Internetware - Internetware ’09, 2009, pp. 1–10.

[20] S. S. Yau, F. Karim, Y. Wang, B. Wand, and S. K. S. Gupta, “Reconfigurable context-sensitive middleware for pervasive computing,” IEEE Pervasive Computing, vol. 1, no. 3, pp. 33–40, Jul. 2002.

[21] L. O. Bonino da Silva Santos, R. P. van Wijnen, and P. Vink, “A service-oriented middleware for context-aware applications,” in Proceedings of the 5th international workshop on Middleware for pervasive and ad-hoc computing held at the ACM/IFIP/USENIX 8th International Middleware Conference - MPAC ’07, 2007, pp. 37–42.

[22] R. P. Pinto, E. Cardozo, and E. G. Guimaraes, “A Component Framework for Context-Awareness,” 2008 International Wireless Communications and Mobile Computing Conference, pp. 315–320, Aug. 2008.

[23] A. Kalaiselvi, V. Indumathi, J. Madhusudanan, and V. P. Venkatesan, “Implementation of Generic Context Middleware for Context-Aware Applications,” International Journal of Engineering Research and Technology (IJERT), vol. 1, no. 3, pp. 1–7, 2012.

[24] T. Strang and C. Linnhoff-popien, “A Context Modeling Survey,” in Proceedings of UbiComp: 1st International Workshop on Advanced Context Modelling, Reasoning and Management, 2004.

[25] B. N. Schilit, N. Adams, and R. Want, “Context-aware computing applications,” in Workshop on Mobile Computing Systems and Applications, 1994, pp. 85–90.

[26] R. Löwe, P. Mandl, and M. Weber, “Context Directory : A Context-Aware Service for Mobile Context-Aware Computing Applications by the Example of Google Android,” in 9th IEEE International Workshop on Managing Ubiquitous Communications and Services, pp. 75–80, March 2012.

[27] “Hibernate,” Red Hat, Inc., 2012. [Online]. Available: http://www.hibernate.org/. [Accessed: 23-Jan-2012].

[28] “Language Guide - Protocol Buffers,” Google Inc., 2012. [Online]. Available: https://developers.google.com/protocol-buffers/docs/proto. [Accessed: 23-Jan-2013].

[29] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton, “Cyberguide: a mobile context-aware tour guide,” Wireless Networks, vol. 3, no. 5, 1997.

[30] R. Want, A. Hopper, V. Falcão, and J. Gibbons, “The active badge location system,” ACM Transactions on Information Systems, vol. 10, no. 1, pp. 91–102, 1992.