Andreas Reinhardt, Johannes Schmitt, Farid Zaid, Parag Mogre, Matthias Kropff, Ralf Steinmetz: *Towards Seamless Binding of Context-aware Services to Ubiquitous Information Sources*. In: Proceedings of the 4th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS-2010), February 2010. ISBN ISBN:978-0-7695-3967-6.

Towards Seamless Binding of Context-aware Services to Ubiquitous Information Sources

(Invited Paper)

Andreas Reinhardt, Johannes Schmitt, Farid Zaid, Parag S. Mogre, Matthias Kropff, Ralf Steinmetz
Multimedia Communications Lab, Technische Universität Darmstadt, Rundeturmstr. 10, 64283 Darmstadt, Germany
{*areinhardt, jschmitt, fzaid, pmogre, mkropff, ralf.steinmetz*}*@kom.tu-darmstadt.de*

***Abstract*—The area of context-aware computing has received much attention in the last decade. However, many systems to determine a user's context are focused on integrating a confined set of sensors into their decision-making algorithms, and lack extensibility for new and novel sources of context information. This limitation however has often resulted in monolithic software systems, reasoning on the user's context from a static set of predefined rules. Therefore, most of these systems can not adapt to the user during runtime, and have not found wide adoption in reality. Unlike the existing approaches, we present a concept for a context-aware system circumventing these limitations by providing generic abstraction layers among its components. We show a concept for an extensible context-aware system with autonomous adaptation to changes in the user's preferences, and provide arguments for our design decisions. After presenting the constraints for all participating entities, including sensors, middleware, and actuation and/or application frontends, we describe ContextFramework.KOM, an example implementation of the proposed concept.**

I. Introduction

The vision of ambient intelligence [1] is based on using ubiquitous devices, integrated in a seamless and unobtrusive manner into the user's environment. Systems mainly aim to support the user, ranging from adaptation to his preferences to the integration with actuation systems. While such intelligent environments allow for a wide range of application scenarios, including ambient assisted living or intelligent buildings, many of today's implemented systems are often either based on static rules (e.g., if the temperature exceeds a given threshold, then turn on the air conditioning), or require a significant amount of human interaction to adapt the desired system behavior to the user's preferences.

Let us e.g. consider an employee P with different tasks to accomplish throughout the day, which he starts with writing a letter. As he must not miss any incoming calls by his superiors, he cannot simply forward all calls to the mailbox, resulting in frequent interruptions by incoming phone calls. During lunch break, he needs to be available on his cell phone in urgent cases, and must hence always keep in mind to set up a call redirection before leaving his office. In the afternoon, he attends a meeting, prior to which he must keep in mind to set his mobile phone to vibration mode. In all of the situations, the tedious process of manual phone setup is imposed on P. *Context-aware* systems are a viable approach to reduce the load on the user when e.g. configured to automatically perform the telephone setup depending on the user's situation.

We term all information that can be used to describe the situation of a user as his *context* in the remainder of this paper (cf. [2] for an elaborate definition). In general, only subsets of the available context information are relevant to classify specific aspects of the user's situation. We refer to these aspects as context *dimensions*, representing characteristic features that can take a set of discrete classes, including the ones exemplary depicted in Fig. 1. The notion of context *classes* describes the user's current state within a certain context dimension. For example, a possible class for the *location* dimension can be *at the workplace*, while the corresponding class for the *task* dimension might be *writing*.

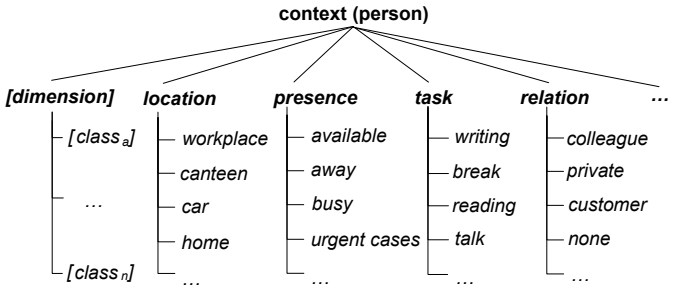

Fig. 1. Examples for different context dimensions of a person

If systems providing ambient intelligence are supplemented by context information, they can base their decisions thereon and thus better adapt to the user's *concept*, i.e. his desired behavior of the system. In the aforementioned example, the user's concept would – amongst others – contain his preferences regarding call forwarding, e.g. to forward all incoming phone calls to his cell phone when he is not at his workplace, or to redirect all incoming calls to the mailbox unless they originate from a superior colleague when the user is busy. Especially as the user's concept is generally hidden and cannot be transferred to an exhaustive set of rules easily, determining the correct classes for a set of context dimensions is inherently difficult. Our contribution to allow for context-aware services is the presentation of a generic, adaptive approach to use context information for various applications. Relying on a set of information sources, comprised of *sensor* and *processor* devices (where processors can perform various, even complex, operations on input parameters), the system must be capable of covering multiple context dimensions and offer easy-to-use interfaces to applications that want to use context information.

After describing an exemplary set of context dimensions and their applications in Sec. II, we list the requirements for any system targeted to determine context classes in Sec. III. A generic approach towards the design of such systems is shown in Sec. IV, and our implementation of the ContextFramework.KOM system provided in Sec. V. We summarize related work in Sec. VI, and conclude this paper in Sec. VII.

II. Selected Context Dimensions

In this section, we present four representative context dimensions and briefly describe information sources that provide corresponding context information as well as application scenarios in which the resulting context classes can be used.

A. Location

A rather basic context dimension is the location of a user or an object. The knowledge about location enables location-aware or location-based services, e.g. automatic rerouting of phone calls to a mobile phone when the user is not at his workplace. Also, other dimensions might depend on location information, as the location might pose limits to the set of possible tasks or available devices. The set of information sources that can determine the location are manyfold, including GPS, WiFi and Bluetooth neighborhood relations. However, also other information sources can be queried to deduce a user's location, e.g. room activity meters or indicators for the computer usage of a given user account.

B. Presence

The context dimension of presence is widely employed in communication systems today, where it is used to display the availability state of participating users. Although users need to set their state manually in most systems (e.g. instant messaging tools), a couple of implementations also rely on a small and fixed set of local information sources (such as the keyboard or mouse activity). Both variants however allow the systems to selectively forward or block incoming communication requests, depending on the users presence state. Taking a more global view on presence leads to a large range of dependencies in terms of information types. Besides information about the current location, at least the available communication devices and the current task are relevant to determine the user's presence.

C. Task

The complexity of capturing information about a user's current task depends on the task itself; while the user is working at his computer, the running applications and data from the input devices (e.g. average mouse movement speed or key stroke rate) can be used to support the determination process of the current task. When the user does not actively use his computer, i.e. in states like *meeting*, *coffee break*, or *reading*, a different set of information sources, including calendar, location, or movement, become relevant. While some tasks may allow the user to be interrupted at all times, only interruptions with a high importance should be forwarded to the user while performing complex or demanding tasks.

D. Relation

Commonly, users have detailed preferences regarding the handling of incoming communication requests. Besides their current task having an influence on their decision whether to accept a request, an even stronger aspect is the social relationship to the enquiring person. While an interruption may not be tolerated when the caller is unknown to the callee, the decision to reject a request might be revoked when some social relation is determined. Knowledge about the relation can e.g. be derived from recent interactions, proximity observation, communication logs, or contact lists. Forwarding calls in a context-aware manner can also reduce the need to carry several communication devices to distinguish between private and business contacts.

III. Requirements for Context-Aware Systems

Any system targeted to determine context states in a generic and adaptive way needs to fulfill a set of requirements. We intentionally present the major challenges in an unsorted list, as their weighting and priority may differ between application scenarios.

It must be open to new information sources during runtime: Anticipating that new sensor types result from the evolution of embedded systems and sensor devices, it is essential to remain extensible to these types and integrate them into the system without reconfiguration efforts.

Extensibility by new context dimensions: New sensor types and application demands might necessitate the definition of new context dimensions. The system should thus also provide extension options for this case.

Only relevant sensors should be queried: Often, a great number of sensors may be present within the system, but only a small subset contributes to the determination of the user's context. The system should thus confine its queries to the relevant information sources to use available bandwidth and energy consciously.

Integration of existing systems: To reduce additional required monetary expenses and integrate with ubiquitous devices like mobile phones more easily, the system should be capable of being run on such devices and platforms. It must therefore provide common interfaces for both information sources and applications that use context information.

Tolerance for failing and missing sensors: Fluctuations of radio links, or device failures may lead to incomplete sensor query results or entirely missing data. The system needs to be able to cope with such situations and react adequately.

Maintaining latency and quality parameters: Obtaining sensor information might be a time-consuming process, if devices are placed at remote locations or connected over slow links. However, incoming context requests should be served within reasonable time, possibly necessitating means to integrate sensors with greater querying delays.

Adaptation to concept drift/shift: In many cases, user concepts evolve over time. To avoid incorrect context classes to be provided by the system, it must be capable to react to gradual (*drift*) and sudden (*shift*) changes of concepts.

Coverage of multiple context dimensions: As presented in the preceding section, multiple dimensions of context exist and might be required to be evaluated simultaneously. The system should therefore provide capabilities to determine more than one context dimension at a time.

Scalability: The system needs to be scalable and must maintain regular operation mode even during periods of high load and a great number of registered sensors.

Privacy: As private information about users is collected and processed within such systems, eavesdropping attacks at both radio links and the processing system might pose risks to the privacy. Ideally, the user should be in control which information he provides at which instant in time.

IV. System Concept

To meet the aforementioned requirements, we propose the use of a modular middleware-based approach, as shown in Fig. 2, which we present in more detail in the following subsections.

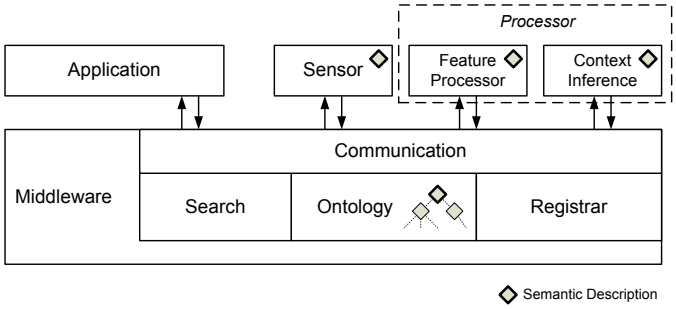

Fig. 2. Modular structure of the context system

A. Information Sources

To avoid compatibility problems when combining several heterogeneous devices and technologies, the system must be capable to integrate them without the need for extensive manual reconfiguration. Especially when a broader set of sensors and especially embedded devices, which may not be connected over wired Ethernet, are part of the system, suitable protocols and network transport mechanisms must be identified and implemented. In general, context dimensions are bound to sensors and processors, which provide information of different *relevance* and *accuracy*. Instantiations of sensors include dedicated physical sensors, either directly connected to a computer, or over wireless radio technologies. However, sensors can also be present as software suppliers, which extract sensor information (such as the keystroke rate or the current open application) from a computer system controlled by the corresponding user. *Processor nodes* might be present in form of databases, mapping the value of an input type to another value of another data type (e.g. GPS coordinates to the according room number). More sophisticated options, such as feature extraction tools, which extract the most relevant information from a greater set of data, or high-level sensors, like image processing tools, can also be integrated as processor nodes.

B. Common description

To allow for new and novel sensors to be interfaced, it is essential for information sources to describe their functionality. Although this description can be limited to the input (in case of processors) and output data types as well as meta information (such as expiry time or accuracy of sensed data), it must be generic enough to allow the integration of heterogeneous sensors, which may range from very small embedded systems up to high-definition image processing server farms. It is obvious that syntactic descriptions are insufficient to determine if sensor readings can be used by a processor component, as they lack information about the *meaning* of the contained information. Instead, the use of semantic annotations is well suited to bridge the gap. A variety of semantic sensor description languages exist, many of them inspired from the area of Web Services [3]. Special emphasis has been put on making service descriptions understandable to machines, so that any application can use many different services, even though it may lack knowledge of the service a priori. Furthermore, the vocabulary of data types must be agreed on at all participating systems. Ideally, a comprehensive and extensible semantic data structure, such as an ontology, is used to also incorporate relations between such types, e.g. inheritance or equality.

C. Data Type Management

Information sources must maintain information about the data types for their input and output parameters. To ensure that devices from different vendors are interoperable and maintain extensibility during system runtime, a static semantic data structure is obviously insufficient. A better solution is to extend the ontology automatically when yet unknown sensor types are encountered. Further means must therefore be integrated to ensure integrity of the ontology, eliminate redundancies, and integrate the correct relations to existing data types. The use of semantic data types obsoletes static system settings or search requests to be adapted to new sensor devices, but instead generates queries based on a sensor's semantic description. Extensibility of the system is thereby ensured while maintaining full functionality. *Loose coupling*, i.e. means to replace information sources by semantically equivalent ones during runtime, is a significant benefit of semantic data type management, and also covers for missing and failing sensors.

D. Communication

To provide an abstraction for the inherent need of communication between heterogeneous systems, the concept of *connectors* can be applied. Representing a generic interface to call methods provided by other systems, any data exchange passes through the connector, which chooses the right means to forward each request. It thus effectively alleviates the need for looking up where another service is running and how it can be invoked. Another advantage of a connector-based architecture is its extensibility; new systems joining the network only need to implement an existing connector to interoperate. The central issue related to the connector-based structure is the

discovery of information sources in the network. Approaches in wired networks usually make use of a centralized directory to manage available services and means of connecting to their host systems. Sacrificing scalability and thus only applicable in small networks, broadcasting can also be used to assist service discovery. As context-aware systems comprise networked devices fitted with physical sensors, parameters like QoS (*Quality of Service*) and QoI (*Quality of Information*) gain importance. Requirements for QoS parameters may arise from time limits on the process of context determination, but QoS also addresses scalability, as large numbers of sensors might be attached to a system. The QoI metric describes the expressiveness of the aggregated information; the challenge is to find *enough relevant* information in limited time to allow the context classification algorithm to make decisions with high accuracy.

E. Search

Even when annotating information sources semantically, information lookup algorithms should be designed in a way to avoid iterating over all available sources of context information for a given dimension, which would result in a great number of requests and possibly lead to context information unrelated the original search request. As this affects both scalability and the quality of the resulting decision, any context determination system should ideally only query relevant sensors. Scalability is hereby inherently ensured by using search algorithms, which can adapt to different applications and to a dynamically changing sensor network topology without the need for a manual redefinition of the search request. As processor nodes always require input from other information sources queried previously, an *iterative search* is mandatory, where results retrieved in previous iterations are used as input parameters to successive requests. To integrate slow and resource-constrained devices into the system, specialized means to access their need to data be integrated. While caching of previous query results may – in combination with meta information about the sensor information quality – reduce the number of queries to the sensor, adaptive pre-fetching of sensor data can be used to retrieve sensor data before an actual query takes place to reduce the delays when polling the sensor. To adapt to the traffic present for sensors and processors, both querying approaches and publish-subscribe architectures can be used to retrieve data in an efficient and timely manner.

F. Context Classification

The context classification mechanism, deducing a user's context state from available context information, is a special type of processor node. To determine correct context states from the set of context information, the system must contain an autonomous classification entity, targeting to approximate the user concept as good as possible with minimum user interaction required. Several approaches towards adapting the classification model to the user's concept are known, including the definition of static rules or the configuration of the system through expert knowledge [4]. Both contradict with the defined requirement of future extensibility, and pose an additional cost, as the entire system configuration must be defined in both cases. A better suited approach therefore is to determine the patterns between the sensor data and the user's behavior from a set of training data. The extracted patterns can then also be used to determine the probable results for new and not yet encountered situations (*extrapolation*). We describe an approach that uses machine learning techniques to deduce context states from available context information in Sec. V-C. Preprocessing of sensor information might also be required in the system, to eliminate noise on sensor readings or set newly encountered observations into relation to existing data.

G. Application

When context-aware systems provide generic interfaces, integrating new applications and user frontends is a simple process, as they merely need to interface the system over one of the available connectors. Examples for basic frontend systems include textual displays, indicating the status of a user to his friends, or a daemon that notifies a person whenever an object's context changes. More sophisticated usage scenarios envisage the integration into workflows like call management or business processes. When corresponding processor services are integrated into the system, they can autonomously trigger actions to establish phone calls or reroute them when the callee is unavailable. Knowledge about the presence state also allows placing a phone call to any person within a group of persons, given the constraint that the person must be available over a landline phone connection. We present a more detailed discussion on arising options to use determined context states through frontends in Sec. II. Context information can also be used in intelligent buildings or intelligent signage, which may even include personalized recommendation systems or actuation over home automation buses (e.g. switching the lights off when the user has left the room).

V. ContextFramework.KOM – an implementation

We have determined the requirements for a context-aware system in the preceding sections. However, to prove the feasibility of our concepts, a real system has been implemented, following the structure shown in Fig. 2, which we describe in detail in the following subsections. For further design decisions and implementation details, we refer the interested reader to our technical report [5].

A. Information Sources

As all information sources need to provide semantic self-descriptions to allow for loose coupling, we have made use of the semantically enriched Web Ontology Language for Services (*OWL-S*) [6] to describe the features and functionalities of all sensing devices. To allow embedded devices with limited capabilities to join the network, we have implemented a lightweight version of the protocol, reduced to the essential set of data required to describe a sensor. New sensors can easily be integrated with the network of information sources by registering their description to the central server instance,

which runs a middleware system. Sensor invocations can be realized over a set of supported protocols, catering for the heterogeneity of the sensor set. Including the XML-RPC [7], RMI [8], and R-OSGi [9] transport mechanisms, the system allows to integrate a wide range of sensors. When integrating nodes in wireless sensor networks, compressed radio messages are employed to efficiently use the available bandwidth and energy budget [10].

B. Middleware

Interpreting sensor and processor functionalities as services yields multiple advantages. Using a service-oriented system architecture provides the required flexibility to cater for failing sensors and dynamic loading of modules during runtime. The OSGi service platform [11] provides the functionality to ease software integration. Services and applications can be integrated and maintained via the network. In our system, the Concierge distribution [12] was selected, since it has a small filesystem and memory footprint while providing all required functionality. The system has been implemented using the Java programming language to allow for portability to many different platforms. Functionalities of the middleware layer can be summarized as sensor management and search functions. The list of registered sensors is kept up-to-date, both in terms of signing up sensors and removing stale sensors from the list to avoid unnecessary calls to non-existing devices. Besides maintaining the ontology of sensor types and their relations, the middleware also provides functionality for searching dedicated sensor types. Search queries can be confined to any subset of sensors to avoid scalability problems and lead to an improved quality of the search result.

C. Search and Classification

The target of adaptive search mechanisms is to determine the relation between a search request and the available set of sensors, and only query sensors which match both the desired quality and relevance. To cater for these options, our implemented system relies on the use of the extensible ontology as well as an iterative search process. When either the desired quality is reached, or the allowed search time is exceeded, results are forwarded to the evaluation process. Evaluation modules for a set of context dimensions have been implemented, and are presented in more detail in Sec. II. We have employed machine learning techniques to support the adaptation of the model, forming the basis for any context decisions, to the user's preferences and wishes. The evaluation instance features an online learning mechanism with gradual forgetting and adaptive window sizing [13]. All results from evaluation modules are fed back into the middleware system as new information sources, allowing their results to be used by all other components of the system, including other processors and frontends.

D. Frontends

The system has been enriched by a set of frontends, which make use of the determined context information in various regards. We have specifically configured a telephony server to only route calls to the callee when he is in an available context state, and configured instant messaging tools to automatically adapt to the context state and avoid pop-up windows when the user is busy. The system has additionally been extended by a set of additional services, including an web based portal for configuration, a control system for sensor management, and a set of non-obstrusive feedback interfaces to support the adaption algorithms.

VI. Related Work

Evolving from Mark Weiser's vision of ubiquious computing [1], many approaches have been made to detect a user's context and react to its changes. Some initial approaches (e.g. [14], [15]) hereby derive contexts from a static set of sensors. The modules for sensor coupling are mostly hardwired in the application code, which generally complicates the application development and poses severe reusability issues.

The CoBra system in contrast supports loose coupling of sensors by employing a central broker for context data [4]. It uses OWL to model context and keep a consistent knowledge, as well as to enforce privacy rules specified by the users. Similarly, the Context Toolkit [16] presents a widget-based context infrastructure which abstracts the low-level sensors from the applications using them and thus supports the reuse of context data by multiple applications. However, the Context Toolkit fails to support important features addressed in this paper like drifting concepts, failing components and fully transparent acquisition of context information. The CALAIS project has been designed with a focus on determining a user's location [17]. In contrast to our proposed concept, CALAIS is limited to a set of static evaluation algorithms and only relies on matching the syntactic descriptions of information sources to pre-defined event templates.

Comparable to our presented approach, SOCAM uses OWL for context representation and knowledge sharing [18]. It also proposes an OSGi-based architecture for provisioning and delivery of context-aware services. The service behavior is driven by reasoning about OWL rules that are specified by the service designer. However, unlike our approach, the end user in SOCAM still lacks the ability to adapt these rules to changing preferences. ContextWare proposes a scheme for late binding of sensors by allowing them to be wrapped as web services [19]. The interface of the service is exposed to the system and semantics are added to the provided data types according to an RDF ontology. However, this last step requires the user interaction to supply the right mapping. In contrast, our approach facilitates extending the ontology at runtime without need for user intervention. The "Suggested Upper Merged Ontology" presented in [20] addresses the challenges of merging multiple existing upper-level ontologies into a single data structure and provides hints on the design of extensible ontologies. In addition, the authors present means towards performing a semantic search, which is closely related to the search mechanisms present in our concept.

VII. Summary

In this paper, we have outlined the features to be supported by any generic, extensible and modular context-aware system, including integration of new sensor types, runtime sensor binding, sensor fault-tolerance, query of relevant sensors and accommodating generic context dimensions. We detailed our concept for supporting these features. In particular, we showed the advantage of using an extensible domain ontology for sensor description, loose coupling and search enhancement. Besides, we proposed a service-oriented design based on the OSGi framework to facilitate platform-independent and efficient interactions. User preferences can be accommodated by adoption of a kind of supervised machine learning which can be adapted through user-friendly feedback frontends. As a proof-of-concept, we presented the implementation of ContextFramework.KOM, which we utilize to capture a wide range of context dimensions.

A. Outlook

We expect that context-awareness is going to be the driving technology for rich personalized services in future. Allowing for augmented reality systems, smart homes/offices, and many further applications, context-aware systems hold the potential to change the way people interact with their environment and experience the world. However, we believe that some issues need to be addressed and considered right from the design phase for future context-aware systems if these are to be successful and accepted by the users.

First and foremost, privacy and security issues need to be addressed for such systems in depth, as private and sensitive data is required to enable personalized services. As we expect the numbers of users and services to increase, strong emphasis must also be put on designing distributed systems to maintain scalability. This corresponding distributed nature of processing and collection of context information implies that privacy and security issues become even more important. Further, when designing context-aware systems, attention should be paid to the design complexity of the solutions to gain broad acceptance among application designers. The computational complexity of the designed solutions will also play an important role; users will expect context-aware services to be provided by an ever increasing set of small and mobile devices, where heavy computational requirements may be infeasible due to processor and battery lifetime limitations. The system design should provide developers with a set of primitives which permit the development and integration of diverse information sources and sensors. The ability to monitor and diagnose faulty behavior in context-aware systems should also be considered, as well as the integration of self-organization and self-healing properties. We feel that the vision of using context-aware systems, being ubiquitous and dramatically changing the way of interacting with the world, can become a reality. However, there are several research challenges which need to be tackled on the way to its realization. We intend to use our presented architecture as a baseline and build up on it to address the issues which we have presented.

Acknowledgment

This research has been supported by the German Federal Ministry of Education and Research (BMBF), and by the German Research Foundation (DFG) within the Research Training Group 1362 “Cooperative, adaptive and responsive monitoring in mixed mode environment”.

References

[1] M. Weiser, “The Computer for the Twenty-First Century,” *Scientific American*, 1991.

[2] A. K. Dey, “Understanding and Using Context,” *Personal and Ubiquitous Computing Journal*, vol. 5, no. 1, 2001.

[3] World Wide Web Consortium, “Web Services Glossary,” Online: http://www.w3.org/TR/ws-gloss, 2004.

[4] H. Chen, “An Intelligent Broker Architecture for Pervasive Context-Aware Systems,” Ph.D. dissertation, University of Maryland, Baltimore County, 2004.

[5] J. Schmitt, M. Kropff, A. Reinhardt, M. Hollick, C. Schäfer, F. Remetter, and R. Steinmetz, “An Extensible Framework for Context-aware Communication Management Using Heterogeneous Sensor Networks,” Multimedia Communications Lab, TU Darmstadt, Tech. Rep. TR-KOM-2008-08, 2008. [Online]. Available: ftp://ftp.kom.tu-darmstadt.de/pub/TR/KOM-TR-2008-08.pdf

[6] World Wide Web Consortium, “OWL-S: Semantic Markup for Web Services,” Online: http://www.w3.org/Submission/OWL-S/, 2004.

[7] D. Winer, “XML-RPC Specification,” Online: http://www.xmlrpc.com/spec, 1999.

[8] A. Wollrath, R. Riggs, and J. Waldo, “A Distributed Object Model for the Java System,” in *Proceedings of the USENIX Conference on Object-Oriented Technologies*, 1996.

[9] J. Rellermeyer, G. Alonso, and T. Roscoe, “R-OSGi: Distributed Applications Through Software Modularization,” in *Proceedings of the ACM/IFIP/USENIX 8th International Middleware Conference (Middleware)*, 2007.

[10] A. Reinhardt, M. Hollick, and R. Steinmetz, “Stream-oriented Lossless Packet Compression in Wireless Sensor Networks,” in *Proceedings of the 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2009.

[11] OSGi Alliance, “OSGi Service Platform Release 4 Specification,” Online: http://www.osgi.org/Specifications, 2007.

[12] J. Rellermeyer and G. Alonso, “Concierge: A Service Platform for Resource-Constrained Devices,” in *Proceedings of the ACM EuroSys*, 2007.

[13] J. Schmitt, M. Hollick, C. Roos, and R. Steinmetz, “Adapting the User Context in Realtime: Tailoring Online Machine Learning Algorithms to Ambient Computing,” *Mobile Networks and Applications*, vol. 13, no. 6, 2008.

[14] K. V. Laerhoven, “Online Adaptive Context Awareness starting from Low-Level Sensors,” Master’s thesis, Free University of Brussels, 1999, http://www.teco.edu/tea/thesis99.ps.

[15] J. Rekimoto, “Tilting Operations for Small Screen Interfaces,” in *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology (UIST)*, 1996.

[16] A. Dey, G. Abowd, and D. Salber, “A Context-based Infrastructure for Smart Environments,” in *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE)*, 1999.

[17] G. J. Nelson, “Context-Aware and Location Systems,” Ph.D. dissertation, University of Cambridge, 1998, http://www.sigmobile.org/phd/1998/theses/nelson.pdf.

[18] T. Gu, H. Pung, and D. Zhang, “Towards an OSGi-based Infrastructure for Context-aware Applications,” *IEEE Pervasive Computing*, vol. 3, no. 4, 2004.

[19] T. Szydlo, R. Szymacha, and K. Zieliński, “Context Generation and Structuralization for Ambient Networks,” in *Proceedings of the 1st International Conference on Autonomic Computing and Communication Systems (Autonomics)*, 2007.

[20] A. Pease, I. Niles, and J. Li, “The Suggested Upper Merged Ontology: A Large Ontology for the SemanticWeb and its Applications,” in *Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, 2002.