A Generalized Context-aware Architecture in Heterogeneous Mobile Computing Environments

Robert Schmohl and Uwe Baumgarten
Department of Informatics
Technische Universität München
D-85475 Garching bei München, Germany
Email: schmohl@in.tum.de, baumgaru@in.tum.de

***Abstract*—The rapid development in the mobile computing domain has significantly expedited the utilization of context-aware systems. Present approaches utilizing awareness of context specialize on their unique domain of employment. Although similarities between those approaches exist, the correspondent concepts and systems are vastly heterogeneous. Another aspect of the rapid evolution of mobile computing is the emergence of a highly heterogeneous spectrum of technologies. This issue complicates the demand for interoperability, which is inherent in mobile computing environments. However, as with context-aware research, most of present approaches addressing heterogeneity are specializing on solving the problem domain-specifically. In this paper, we present a conceptualization on generalizing both context-awareness and heterogeneity-handling in mobile computing environments. Concluding, we derive a general architecture, which overcomes the heterogeneity issues present in both context-awareness and interoperability domains.**

I. Introduction

The rapid evolution of mobile computing has had a concurrent effect on related research domains such as context-aware computing [1]. This has lead to a vast amount of experimental systems being employed for a large spectrum of use cases. Due to the individual focus of each approach on its target domain, naturally, all of the approaches make the current state of context-aware computing appear heterogeneous.

Furthermore, mobile computing is characterized by a high level of heterogeneity itself, since standards for the exchange of information (communication protocols, workflows,information display, etc.) are either not existent or not necessarily commonly obeyed by device manufacturers, software developers and network providers [1]. This aspect is especially reflected by heterogeneous characteristics of mobile devices, operating system, resources and network capabilities [2].

Our goal is to overcome those heterogeneity issues in the domain of context-aware computing. We have chosen to address the problem with two different approaches:

- *Generalizing context-aware systems:* Our focus in this approach is set on identifying common concepts in context-aware computing approaches, which allows us to devise a general concept for context-aware system development.
- *Generalizing heterogeneity-handling techniques:* The goal of mobile computing suggests including devices spanning the entire hardware spectrum [3]. This requirement includes the appliance of various use cases, which are usually handled individually. Hence, we focus on identifying common heterogeneity-handling techniques to derive a general architecture enabling general interoperability.

Both approaches excel the effort to generalize the correspondent sub-domains, which we consider relevant for our objective. In this paper, we document both of the stated approaches to inferentially derive a concept unifying both.

The rest of the paper is structured as follows: Section II outlines the work of related research groups. In section III we document our effort to unify various concepts excelling awareness of context and depict a general architecture. Section IV discusses techniques on handling heterogeneity aspects in mobile computing deriving a generalization as well. Consecutively, we reason an abstract architectural concept enabling context-awareness in heterogeneous environments in section V. Section VI summarizes the work documented here and gives an outlook on our future work.

II. Related Work

Our research documented in this paper is based on the research conducted by groups, which are active in the same domains of context-awareness and heterogeneity-handling. The authors of [4], [5] and [6] have conducted surveys in the context-awareness domain, whereas the work documented in [2] and [3] describes middleware approaches for this area of research. The authors of [7] have reasoned a well formed generic workflow for context-aware computing.

Regarding research focussing on heterogeneity-handling solutions, we have identified two groups focussing on related topics. The authors of [8] present a framework bridging the communication between heterogeneous devices, whereas [9] presents a system capable of integrating heterogeneous devices dynamically and enabling interoperability between those.

III. A Generic Context-aware Architecture

As stated in the introduction, this section summarizes our previous work by shortly surveying the area of context-aware computing and describing an abstract architecture, which excels the common characteristics in context-aware computing [10].

A. Context-aware Computing

Context-awareness is the ability of capturing and processing contexts. Context comprises of contextual information which may be retrieved from heterogeneous sources [4] and is defined as any information to characterize situation of an entity (place, person, object). Concluding, context is a set of the associated situations and actions [7], [4] characterizing the physical surrounding of a device and captured by sensors of the device or the infrastructure.

Context is represented in *context models*. Those are sophisticated data structures, which are populated by abstracting and representing contextual information for further processing. Hence, context modeling is a technique focusing on how to find and relate contextual information, that captures the observation of certain worlds of interest [4]. Context can be represented in various formats ranging from simple key-value-models to graphical representations [5]. Out of those, current research indicates, that ontologies are the most expressive context representation models [2], [5]. Ontologies provide a powerful paradigm for context modeling offering rich expressiveness and supporting the dynamic aspects of context awareness. Ontologies represent *concepts* and *relationships* between concepts, where abstractions from the real world are usually mapped to concepts with relations interconnecting those concepts according to their real-world equivalents' relations.

The composition of context-aware systems is characterized by a high degree of heterogeneity. However, all approaches agree in decoupling context capturing and context processing from application composition [7] by encapsulating the context management logic into middlewares.

Architectures of context-aware systems are decomposed into static and dynamic aspects. Static aspects describe the components in use whilst the workflow of acquiring and processing context depicts its dynamic facets. Both heavily depend on the application domain of each individual system.

An analysis of the architectures of multiple context-aware systems [10] identifies the following abstract components shared by the majority of those approaches:

- *Context sensors* acquire raw contextual data by either sensing the physical environment (hardware) or providing data from other context sources (software) [11], [4]. The primary task of both sensor types is the acquisition of raw data for further refinement into contextual information.
- *Context capturing interfaces* refine the raw data acquired by sensors, which is usually uncertain and difficult to interpret by high-level components [5], into data structures, that are utilizable for higher application levels, thus providing a proper interface for the sensed environments [6].
- *Context repositories* store the current context utilizing context models.
- *Context reasoning components* are responsible for inferencing new context based on the current contextual information in the context repository and new contextual data acquired through the context capturing interface. This process is put in practice by *inference engines*, that work on the basis of rules [12], [11].
- *Context APIs* provide interfaces for context-aware applications to actually utilize contextual information [2].

The enumeration of components shows a flat snapshot of a generic context-aware system. Those may be abstracted into a hierarchy of layers, with each layer representing information on a specific level of detail [7]:

- *lexical level:* Signals from sensors are abstracted into basic context events.
- *syntactical level:* Context events are translated to atomic context information, such as matching sensor data to real-world-properties.
- *reasoning level:* Basic context information is refined and organized, context information is fused into a reasonable representation suitable for more sophisticated processing.
- *planning level:* Context is evaluated, changes in context are detected, reactions to context changes are planned and scheduled.
- *interaction level:* reactions to context changes are executed in form of personal and collaborative interactions with the user and other hosts

With the component representing rather static architectural aspects, the workflow of capturing, storing and utilizing contextual information describes a dynamic aspect by extracting the components' tasks and putting those in sequence reasonably [7], [13], [2]:

1) *Context sensing*: Detection and representation of contextual information [13] by acquiring raw sensor data and refining them into data structures to provide a higher level of context.
2) *Context update and management*: The refined and well presented contextual information is fetched and merged into the context repository. The overall context is updated in the process.
3) *Application of context*: The most current contextual information is fetched from the context repository through the context API to be used by the context-aware application. Important aspects of context application are adaptation to the current context, contextual resource discovery and context augmentation.

B. A Generalized Architecture

In this section, we summarize the common characteristics in context-aware computing and derive a general architecture for context-aware systems.

Figure 1 visualizes this concept employing all of the relevant components layered in levels of abstraction as discussed earlier.

The core of each system excelling awareness of context is the context model storing the current contextual information. As we have stated earlier, ontologies are the first choice made by most research groups due to their high degree of expressiveness. Since the employment of ontologies requires

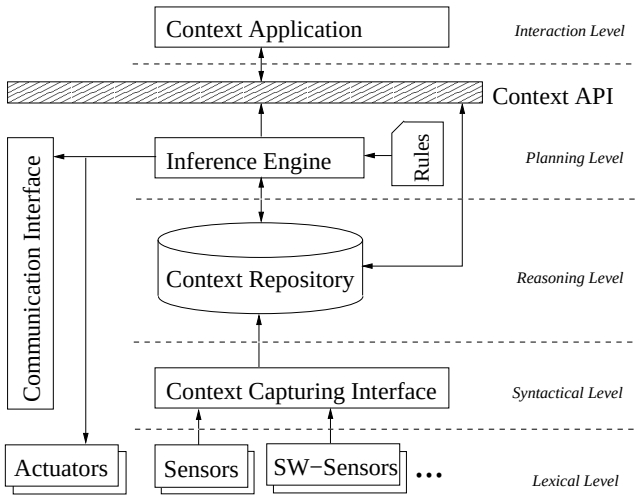

Fig. 1. General Architecture for context-aware Middleware

adequate hardware capabilities, one may be forced to fall back on less demanding context models if the available hardware is constrained.

The workflow of recognizing, updating and utilizing context is strongly inspired by the procedure described previously. *Sensors* (which may explicitly include software components as well) acquire raw environmental data, which is refined by the *context capturing interface* into contextual information. The raw data is abstracted into discrete data structures so that it can be processed further by software at all. The contextual information is then committed to the *context repository*, which is responsible for the persistent storage of the context using an appropriate context model. The context repository controls the access to the contextual data and therefore functions as an interface to both the rest of the context-aware system and any user applications utilizing the context. The manual of altering the context is encoded into *inference rules*, which are enforced by the *inference engine*. Both the rules and contextual information are loaded by the inference engine, which subsequently updates the context in the repository according to the inference rules. Since we are dealing with a distributed system, contextual updates may be propagated to other nodes via the *communication interface*, which is attached to the communication hardware. The inference engine may also trigger any *actuators*, that are affected by the context update. The *context API* provides access to the context for user applications. It represents the architectural cut between context management and context utilization, as we have argued earlier [7]. The context API has direct access to the context repository, meaning it reads the current context and commits user updates into the context. The inference engine may also notify user applications through the context API, if inferencing new context requires it.

In summary, the context reasoned by this architecture remains unchanged until it is altered by a *contextual update*, which may originate from one of the following sources:

- *Environment*: Contextual information is gathered by the sensors.
- *Inference Engine*: New context is derived independently by the inference engine using its appropriate rules.
- *User*: The user commits data to the context through the context API.

IV. The Heterogeneity Issue

Another issue in the context-aware computing domain is the impact of device heterogeneity, which hinders interoperability among mobile devices. This section analyzes this issue and derives an abstract heterogeneity-aware middleware.

A. Heterogeneity Abstraction

In mobile environments, the problem of heterogeneity concerns a wide range of architectural domains. A simple cut allows the abstraction of heterogeneity into 3 different views [2]:

- *Hardware heterogeneity*: Hardware heterogeneity reflects the presence of different devices with different capabilities, as well as different network technologies integrating those devices.
- *Software heterogeneity*: Software heterogeneity is characterized by the presence of different applications and operating systems.
- *Architectural heterogeneity*: This heterogeneity aspect illustrates environments where network interconnections do not share any common architectural characteristics.

B. Heterogeneity Handling

This section focusses on how to overcome the heterogeneity issues in order to maximize interoperability. A key concept in approaching the problem is the design of middleware solutions, which are intended to make heterogeneity transparent. In this section, we describe relevant approaching addressing the issue.

1) Representation of Heterogeneous Data: Heterogeneity defines itself by the presence of different capabilities in the same domains. In order to accomplish interoperability among heterogeneous devices, the first step is often to capture all of their capabilities and structure them into suitable representations. Those *device capability databases* (DCDBs) serve as a data basis for solutions adopting to device heterogeneity.

Whilst DCDBs enable uniform *storage* of heterogeneous data, this principle can be applied analogously to facilitate *workflows* of middlewares dealing with heterogeneous environments. In that case, the middleware uses uniform *intermediary representations* of data [8], [14]. Such representation represents a generalization of all supported communication semantics from the heterogeneous spectrum. When communicating with heterogeneous partners the data is dynamically adapted to the target's specification at runtime using DCDBs as described previously.

2) Common Interfaces: One of the most basic approaches in enabling interoperability among mobile hosts is to identify their common interfaces and exploit them accordingly. Those interfaces allow the devices to be very different by hiding their individual heterogeneous characteristics behind their commonly shared interface specification and thus making themselves transparent to each other [9]. This principle is applicable to both domains of hardware (common interfaces such as 802.11) and software (common APIs, such as J2ME).

3) Individual Adaptation: Direct communication among two heterogeneous partners requires, that at least one of the communication partners adapts to the other's communication technique. Extending this approach on a setting with devices implementing numerous different communication techniques and hence enabling uniform communication results in the demand of general interoperability [8].

The following workflow enables this principle [14]:

1) As soon as the communication partner is identified, it is evident which communication standard is needed. Identification is conducted using a device detection mechanism exploiting a device database, which also contains the capabilities of the identified device.
2) Since the communication partner supports only one of many communication standards, an intermediary layer is inserted between the communicating entities for reasons explained in section IV-B1. Incoming and outgoing communication is translated to the intermediary representation first before being committed to the communication partner or the adapting device.

C. A Generalized Architecture

After having presented techniques how to address heterogeneity aspects, which are present in mobile computing environments, we can derive a general architecture implementing those approaches as illustrated in figure 2.

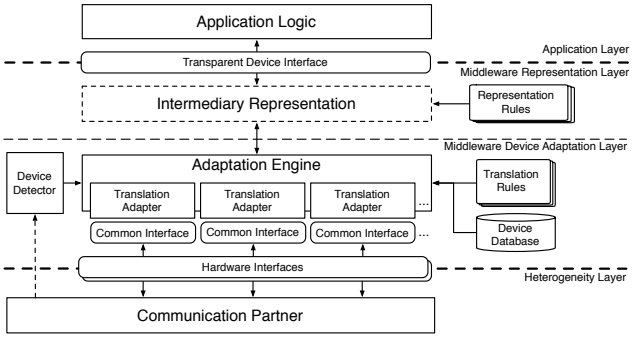

Fig. 2. General Architecture for heterogeneity-aware Middleware

The workflow to enable interoperability among heterogeneous devices and systems basically utilizes the techniques depicted earlier in section IV-B. First, the *device detector* identifies the partner device by matching the device's identification attribute with the corresponding entry in the *device database*. With this information available the middleware is able to negotiate communication between the *heterogeneous device* on the heterogeneity layer and the *parent application* on the application layer, as illustrated in figure 2. On the top, it provides a *transparent device interface* to applications, hiding all heterogeneity issues from them. On the bottom, it utilizes native hardware interfaces to connect to the communication partner. The middleware decomposes into 2 layers:

- *Device adaptation:* We have labeled the core component on this level the *adaptation engine*, which adapts all communication to the heterogeneous device. Each of its translation adapters is bound to a common interface present on the deployment system. It utilizes the device information from the device database to query information on how to actually adapt the communication to the partner device. For reasons of modularity, the translation rules are kept separately and interpreted at runtime. This approach allows the adaptation engine to handle heterogeneous devices through the common interfaces (bottom) and to provide a device-independent interface to the representation layer (top).
- *Data representation:* This layer's purpose is to make all information send through the middleware uniform and device-independent. It therefore uses an intermediate representation format.

The architecture presented here allows transparent communication with heterogeneous devices. The application logic depicted on top of the transparent device interface in figure 2 may be instantiated as applications in many domains, including the following exemplary selection:

- *Communication:* Heterogeneous devices communicate seamlessly with each other without caring for syntax, semantics or protocols.
- *Services:* Services may be offered to a wide spectrum of heterogeneous devices allowing service providers to extend the range of their service, meaning the increase of the amount of potential service users.
- *Interoperability:* Collaborative systems may include heterogeneous devices extending their range of potentially reachable devices.

All of those and any other application cases may profit from neglecting heterogeneity issues addressed by the middleware.

V. Context-awareness in heterogeneous Environments

After having identified common approaches in context-aware computing and heterogeneity-handling in sections III and IV, we aim at fusing the derived generalized architectures together to a single concept. We have chosen to pursue a top-down approach by first designing a high-level system sketch, which we concretize subsequently.

From the scrutinized domains of context-aware computing and heterogeneity-handling, we have learned, that concerned architectures are layered providing distinct interfaces between major layers. Those boundaries denote major abstraction cuts and help us to eventually derive our concept. From both

domains, we have identified a total of 3 abstract interfaces, which we consider crucial for the construction of a context-aware and device-independent middleware:

- *Context-API:* providing access to the most current contextual information, hence offering an interface for application utilizing context.
- *Transparent Device Interface (TDI):* enables device-independency, thus leveraging heterogeneity-transparency for all components accessing this interface.
- *Hardware Interfaces:* provide access to the hardware of the system of deployment.

Further, we decompose our generic middleware concept into two separate system parts handling *context-awareness* and *device-adaptation* respectively. The combination of this 2-part system composition with the interface specification allows us to outline a first architectural sketch. The context-aware system part provides the contextual information through the context API on the top and accesses the heterogeneous environment via the TDI at the bottom. The latter is provided by the heterogeneity-aware middleware part, situated on the lower level and handling device-dependent heterogeneity issues utilizing the hardware interfaces. Hence, the TDI is the architectural cut in the system between the part handling device-dependencies (heterogeneity) on the bottom and enabling device-transparency at the top. Figure 3 depicts this component composition embedded into the described interface scheme.

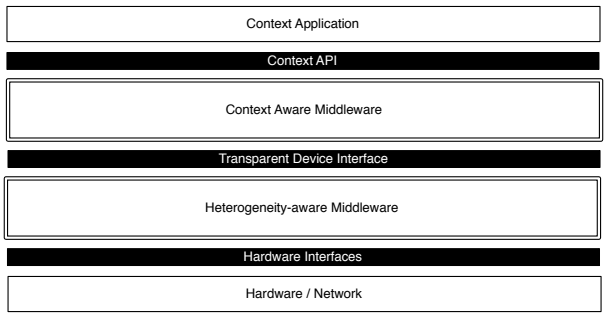

Fig. 3. Middleware combination

This architectural draft allows us to concretize the two individual middleware components in two steps:

1) First we replace the placeholders for context-awareness and heterogeneity-handling with the corresponding general architectures depicted in figures 1 and 2.
2) Then we project the layering depicted in section III-A onto the architecture and extend it accordingly.

The resultant architecture is illustrated in figure 4. Both the context-aware system part and the heterogeneity-handling component group roughly function as described earlier in sections III-B and IV-C.

In order to achieve seamless integration of both middleware components, their merging into the final architecture has yielded the following layering:

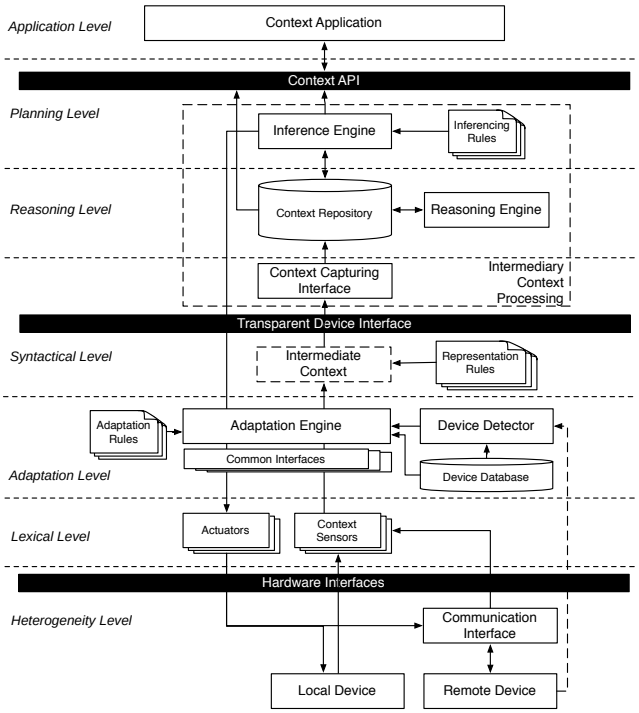

Fig. 4. General context-aware Middleware in heterogeneous Environments

- *Heterogeneity level:* The lowest level denotes the heterogeneous environment of mobile devices, networks and back-end systems. At this level, we also distinguish the view on the local device and remote devices.
- *Lexical level:* Since actuators and sensors of the context-aware architecture are device-dependent, they have been decoupled from its parent middleware and embedded into the heterogeneity-handling part. They provide the system with raw and highly heterogeneous data for further processing.
- *Adaptation level:* This level includes the adaptation engine adapting all communication to each heterogeneous target device. It works bidirectionally interpreting heterogeneous context data from the context sensors and committing contextual updates from the inference engine to the heterogeneous mobile environment.
- *Syntactical level:* The transparent device interface (TDI) hiding the heterogeneity from the rest of the system (from upper side) is located on this level. The device-independent, intermediary representation of the context, which is used by the adaptation engine (as discussed in section IV-C), is located just below this interface. The context capturing interface on the upper side of the TDI refines this raw context information into semantically enriched context (as discussed in section III-B).
- *Reasoning level & planning level:* As depicted in section III-B, context is reasoned and stored in the context repository on the reasoning level whereas the inference engine derives new context on the planning level. Contextual

updates are committed to the adapation engine to be committed to local actuators or the network. The context API provides device-transparent access to the current context.

- *Application level:* Applications accessing the context API may exploit heterogeneity-transparent contextual information and implement their own application-specific business logic. This is the level where contextual interaction takes place and where context is augmented with auxiliary information.

To complement our discussion, we observe uniform context processing spanning from the syntactical level to the planning layer. Therefore, it may be defined *intermediate*, since it is not feasible for any application but the middleware discussed here.

VI. Conclusion and Future Work

In this paper, we have surveyed approaches to implement context-awareness and to handle heterogeneity in mobile computing environments. We have identified common characteristics in each domain and derived those to general architectures. Finally, we have integrated both architectures proposing an architectural concept for a middleware supporting context-awareness in heterogeneous mobile computing environments.

The concept developed in this paper allows a broad look on general mechanisms addressing heterogeneous context-aware computing. It may serve as a base pattern for the development of further systems aiming at specialized use cases in this application domain. The concepts described here may then be adjusted to serve the according use cases.

With the conceptualization presented here our next steps encompass its refinement on lower design levels to analyze the feasibility of our concept and to prepare the construction of a prototype framework implementing this concept. This includes (but is not limited to):

- Deployment issues of the middleware focussing on whether to place it on mobile devices and/or or back-end systems
- Evaluation of distinct domains of context-awareness, especially location and location-based services
- Identification of suitable device detection mechanisms in heterogeneous environments, since heterogeneous devices can hardly identify themselves in a common way
- Analysis of technologies suitable for the component of the presented architecture to evaluate its interaction capabilities with regard to our architectural sketch

We are going to consider various target platforms to enable general applicability of this framework. The platforms in question are yet to be defined, but will likely include representatives spread on a large number of mobile devices and back-end systems.

References

[1] Robert Schmohl and Uwe Baumgarten. Mobile services based on client-server or p2p architectures facing issues of context-awareness and heterogeneous environments. In *PDPTA '07: Proceedings of the 2007 international conference on parallel and distributed processing techniques and applications*, pages 578–584. CSREA Press, 2007.

[2] Ricardo Couto A. da Rocha and Markus Endler. Evolutionary and efficient context management in heterogeneous environments. In *MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–7, New York, NY, USA, 2005. ACM Press.

[3] Martin Modahl, Ilya Bagrak, Matthew Wolenetz, Phillip Hutto, and Umakishore Ramachandran. Mediabroker: An architecture for pervasive computing. *percom*, 00:253, 2004.

[4] Christos B. Anagnostopoulos, Athanasios Tsounis, and Stathes Hadjiefthymiades. Context awareness in mobile computing environments. *Wirel. Pers. Commun.*, 42(3):445–464, 2007.

[5] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Proceedings of the Workshop on Advanced Context Modelling, Reasoning and Management associated with the 6th International Conference on Ubiquitous Computing (UbiComp), Nottingham.*, 2004.

[6] Hans W. Gellersen, Albercht Schmidt, and Michael Beigl. Multi-sensor context-awareness in mobile devices and smart artifacts. *Mob. Netw. Appl.*, 7(5):341–351, 2002.

[7] Eleni Christopoulou, Christos Goumopoulos, and Achilles Kameas. An ontology-based context management and reasoning process for ubicomp applications. In *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*, pages 265–270, New York, NY, USA, 2005. ACM Press.

[8] Jin Nakazawa, H. Tokuda, W.K. Edwards, and U. Ramachandran. A bridging framework for universal interoperability in pervasive systems. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 3–3, 2006.

[9] Christian Bartelt, Thomas Fischer, Dirk Niebuhr, Andreas Rausch, Franz Seidl, and Marcus Trapp. Dynamic integration of heterogeneous mobile devices. In *DEAS '05: Proceedings of the 2005 workshop on Design and evolution of autonomic application software*, pages 1–7, New York, NY, USA, 2005. ACM.

[10] Robert Schmohl and Uwe Baumgarten. Context-aware computing: a survey preparing a generalized approach. In *IMECS 2008: Proceedings of the International MultiConference of Engineers and Computer Scientists 2008*. International Association of Engineers, 2008.

[11] Carsten Jacob, David Linner, Ilja Radusch, and Stephan Steglich. Loosely coupled and context-aware service provision incorporating the quality of rules. In *ICOMP 07: Proceedings of the 2007 International Conference on Internet Computing*. CSREA Press, 2007.

[12] Gregory Biegel and Vinny Cahill. A framework for developing mobile, context-aware applications. *percom*, 00:361, 2004.

[13] Teddy Mantoro and Chris Johnson. Location history in a low-cost context awareness environment. In *ACSW Frontiers '03: Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*, pages 153–158, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[14] Robert Schmohl, Uwe Baumgarten, and Lars Koethner. Content adaptation for heterogeneous mobile devices using web-based mobile services. In *Proceedings of the 5th International Conference in Computing and Multimedia (MoMM2007)*, pages 77–86. Oesterreichische Computergesellschaft, 2007.