Context-aware Computing: a Survey Preparing a Generalized Approach

Robert Schmohl, Uwe Baumgarten *

Abstract—**Present approaches utilizing awareness of context specialize on their unique domain of employment. Although similarities between those approaches exist, the concepts and systems utilized in this context are vastly heterogeneous. Our goal is the abstraction of the current situation in context-aware computing and the development of generalized concepts for approaching the development of context-aware systems. We put our focus on ubiquitous computing and mobile services, since those are especially suitable for employing awareness of contextual information. This paper presents the first step in this endeavor, consisting of an in-depth survey of context-aware computing and the presentation of our future plans to proceed in this project.**

Keywords: context awareness, context models, context management systems, mobile services, heterogeneity

1 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. Nevertheless, most approaches express a certain amount of common characteristics.

Our focus is set on identifying common concepts in context-aware computing approaches, which allows us to devise a general concept for context-aware system development. This paper presents a snapshot of the current state of context-aware computing, which we plan to use as a foundation for further pursuing our objective. We have analyzed several concepts excelling awareness of context and we have consecutively identified their common characteristics. The result of this analysis, which we present to conclude this paper, form the base for the conceptualization of generalizing context aware computing approaches.

*Technische Universität München, Department of Informatics, 85478 Garching, Germany, schmohl@in.tum.de, baumgaru@in.tum.de

The rest of this paper is structured as follows: section 2 introduces the definitions of context. Section 3 discusses how contextual information can be abstracted and represented by context models. In section 4, we summarize architectural characteristics common in most context management approaches. Section 5 concludes this paper by constructing a general architecture for context management systems based on the survey presented here, and it gives an outlook on our future work.

2 Context Definition

Context awareness is the ability of capturing and processing contexts. Context comprises of contextual information which may be retrieved from heterogeneous sources [2]. Concluding, context may be defined as follows:

- Generally speaking. context is any situation to characterize situation of an entity (place, person, object) relevant to the integration of user and application, including the user and application themselves. Concluding, context is a set of the associated situations and actions [3, 2]
- Practically, context is the physical surrounding of a device, which is captured by sensors of the device or the infrastructure. A device's awareness of context may further be categorized as either *direct*, denoting the context is captured by the device's sensors, or *indirect*, in case context is acquired by the infrastructure [4].

3 Context Modeling

Context modeling is the process of abstracting and representing contextual information for further processing. According to [2], this particularly includes the following aspects:

- identification of the most appropriate contextual information, that can model well enough the specific context in a certain domain.
- identification and modeling of relations among pieces of contextual information

- identification of possible dynamic changes in contextual information and thus, modeling appropriate reactions to such changes

Hence, modeling context is a technique focusing on how to find and relate contextual information, that captures the observation of certain worlds of interest [2]. It decomposes into 2 subsequent phases: First, characteristics from real world are abstracted conceptually. Afterward, this concept is mapped on a context model representing the contextual information. We are about to discuss both phases in the subsequent subsections 3.1 and 3.2.

3.1 Conceptual Approach

Context modeling approaches can be classified into two, not necessarily disjoint, taxonomies [2]:

- *Context Theoretic Modeling*: This approach aims at representing context primarily by fusing information describing *situations* of entities, that are dynamically changing by the occurrence of *actions* affecting those entities and thus their situations. There are two alternatives in the context theoretical modeling approach. On the one hand, context may either be described situation-centric as he composition of all entities' situations derived from events/activities involving those entities. On the other hand, the activities themselves can be used to describe situational context, by applying probabilities to the activities of of being performed.
- *Context Conceptual Modeling*: This type of context modeling describes context as *concepts* and the relations among such concepts. Furthermore, such type of modeling categorizes context according to its prevalent characteristics. It subdivides into 2 closely related alternatives. For one, context is described as a conceptual graph, where concepts and its interrelations are directly mapped to a graph's vertices (concepts) and edges (relations). This approach can be refined into an alternative, where sets of prepositions describe context. Similar to the direct approach, the statements of propositional languages are visualized as a graph leaving it to be interpreted as a conceptual graph of semantics.

3.2 Context Models

In order to create a context model, the conceptual model from section 3.1 has to be implemented appropriately by abstracting the conceptual model into structured information. Our survey has resulted in identifying the following set of context models [5, 6]:

- *Key-value models*: Those are the most simple data structures associating context attributes with specific values of contextual information.
- *Markup scheme models*: These models consist of hierarchical data structures based on markup tags including attributes and comments. They are usually implemented as derivatives of SGML [7]. In some cases, markup scheme models are used to describe context as extensions of Composite Capabilities / Preferences Profile (CC/PP) [8] and User Agent Profile (UAProf) [9] to cover the high dynamics of contextual information.
- *Graphical models*: A quite intuitive approach to model context is to represent contextual entities and their relationships graphically. the most prominent examples are Unified Modeling Language (UML) [10], which is a suitable due to its generic structure paired with a strong graphical component; and Object-Role Modeling (ORM) [11], which can be nicely utilized to represent context graphically by identifying facts and enriching those with types and roles.
- *Object-oriented-models* consist of encapsulating contextual information into objects. The information can only be accessed through well defined interfaces and is therefore hidden to from other objects. Due to the nature of object-oriented modeling this approach emphasizes reusability and controlled access to contextual information.
- *Logic-based models* represent a highly formal modeling approach. It is based on logics, which define conditions on which concluding expressions or facts may be derived from sets of other expressions or facts. Those conditions are described by rules in a formal system, so that the facts, expression and rules put all together define the context.
- *Ontology-based models* use ontologies, which are used to represent concepts and relations between concepts. They represent a uniform way for specifying the model's core concepts as well as subconcepts and facts, thus enabling contextual knowledge sharing and reuse.

Current research indicates, that ontologies are the most expressive context representation models [12, 6]. Ontologies provide a powerful paradigm for context modeling offering rich expressiveness and the supporting the dynamic aspects of context awareness. However, they require ontology engines managing the ontologies in use. Those engines generally have high requirements on resources, requiring the employing architecture to support those. This may have negative performance impacts on local context processing, where resource-constrained devices are employed [12].

As stated, ontologies represent *concepts* and *relationships* between concepts. Abstractions from the real world are usually mapped to concepts with relations interconnecting those concepts according to their real-world equivalents' relations [13, 14].

The ontologies representing individual entities must often be implemented in a certain frame to function properly. This frame is the same for all entities implying the need of ontologies being reusable. For this reason, ontology models often consist of two components[3, 14, 13]:

- *General ontology*: Those ontologies represent the general part of the model, which is used by all instances. It applies for all entities as it is. General ontologies are therefore used as a frame for ontologies representing entities.
- *Specific ontologies*: Those are employed entity-specific. Each entity may have unique characteristics, that are represented by its own specific ontology only.

Beside their descriptive purposes, ontologies define sets of rules, that are utilized by the inference engines using the context model to calculate the current context [3].

3.3 Further Aspects of Context Modeling

An interesting aspect of context awareness is the utilization of *distance* of contextual relevance as presented by Roman et al. [15]. This approach conceptualizes context to be distributed on all participating nodes in a network. But a single node in the network is usually not interested in the complete context in the network, but rather only in relevant contextual information in its vicinity, which consist of the contextual data available at the proximate nodes. Thus, each node may define its *context boundary*, depending on its individual preferences. This boundary is therefore individually rooted at the respective node defining its relevant context.

Another important aspect to be mentioned is the the *heterogeneity* of mobile environments, which has a peripheral impact on context modeling. Especially the following aspects are to be regarded [12]:

- *Software heterogeneity*: Context models may have to address certain application requirements. Hence, the presence of heterogeneous software may have to be considered. Additional problems arise when heterogeneous context models are to be employed simultaneously [16].
- *Architectural heterogeneity*: Context models may be bound to specific network domains, because only some context information may be relevant. Multiple heterogeneous networks may have to be considered.

Biegel et al. [17] propose an interesting way of handling *probabilistic aspects* of context modeling. Their context-aware system employs a context model, which encapsulates a situational context into context hierarchies, which represent sets of actions by means of generalization and specialization (e.g. moving -¿ running). Those hierarchies define the actions possible in such situations. To enrich the context with the probability of those action to be taken, the activities in the hierarchy are represented by *Bayesian networks* [18], that are constructed from context sensors. Those networks allow a probabilistic conclusion on the actions represented in the context hierarchy.

4 Architectures of context-aware Systems

In this section we focus on how context aware systems are generally implemented. We especially emphasize architectural issues of component composition and the workflow of context management systems.

4.1 Components

The composition of context-aware systems is is characterized by a high degree of heterogeneity. However, all approaches agree in decoupling context capturing and context processing from application composition [3] by encapsulating the context management logic into middlewares.

An analysis of the architectures of multiple context-aware systems presented in various publications [4, 19, 17, 3, 12, 20] identifies the following abstract components shared by the majority of those approaches:

- *Context sensors*: Those sensors exist either as pieces of hardware sensing the physical environment, or as software component providing data from other context sources [19, 2]. The primary task of both sensor types is the acquisition of raw data for further refinement into contextual information. Concluding, we can depict the following types of sensors [4]: (1) visual and auditory, (2) location, (3) environmental sensors such as motion detectors or thermostats, and (4) software sensors, i.e. software components, that draw information from context sources other than the environment.
- *Context capturing interface*: Data acquired by sensors is usually uncertain and difficult to interpret by high-level components [6]. For this reason, sensor data needs to be refined for further processing by deriving a higher level of context from uncertain multi-modal sensor data - a process, that may be called *sensor fusion* [17]. It is characterized by refining raw sensor data into data structures, that are

utilizable for higher application levels, thus providing a proper interface for the sensed environments [4]. Such data structures may reach from simple data types to sophisticated mechanisms, such as Bayesian networks [18] for probability calculations [17].

- *Context repository*: The context repository stores the current context. It consists of the according data structures used to represent the context model.
- *Context reasoning*: The context reasoning component is 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 [17, 19]. They fetch the contextual information from the repository and the contextual updates from the context capturing interfaces as input parameters, and they output an updated context model, that is committed to the context repository. Rules can be either of global or local scope. Global rules affect the entire context model, local rules however have an effect on local entities only [3].
- *Context API*: The context API provides an interface for context-aware applications to actually utilize contextual information [12]. The employment of such an interface implements the commonly accepted paradigm of separating context management from application logic [3].
- *Context application*: Applications accessing the context API can actually *use* the contextual information for their application-specific purposes. This includes the automatic execution of services based triggered by special contextual conditions, as well as the discovery and allocation of resources relevant to the current context [21]. The aspect of context application especially emphasizes the deployment of legacy applications [19].
- *Communication interface*: Since a context-aware system is distributed in nature, communication needs to be handled appropriately. This especially concerns the reasoning mechanisms, which may want to commit contextual updates into the network, and context capturing components, that may request contextual information from other nodes in the network.
- *Actuators*: Actuators are the counterparts of sensors, that may be utilized as a result of a contextual update [17].

4.2 Layered Architecture

In addition to a flat representation of a generic context-aware system (as presented in section 4.1), such may be abstracted into a hierarchy of layers, with each layer representing information on a specific level of detail [3]:

- *lexical level:* signals from sensors are abstracted into basic context events
- *syntactical/representation 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

4.3 Workflow

So far, we have put our focus on the static aspects of context-aware systems only. To describe the dynamic behavior of such a system we can derive the workflow of capturing, storing and utilizing contextual information from the static architecture from sections 4.1 and 4.2 by extracting the components' tasks and putting those in sequence reasonably.

1. *Context sensing*: Detection and representation of contextual information [21]
   (a) *Acquisition of sensor data (low-level context)*: Raw data is captured by sensors, representing contextual information on the lowest level of abstraction (sensors on the lexical level) [3].
   (b) *Refinement of sensor data*: Raw data acquired by sensors is interpreted and represented in data structures to provide a higher level of context (context capturing interface on the syntactical level) [3].
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 (context reasoning and context repository on the reasoning level) [3].
3. *Application of context*: The most current contextual information is fetched from the context repository through the context API [12] to be used by the context-aware application. This may occur by both pushing or pulling the information to the application

level, dependent on the implementation of the application. The use of the current context includes the following aspects:

- *Contextual adaptation*: Context-aware applications automatically adapt to the current context by updating/executing their according services and their internal states [21]. (context application on the planning level [3]).
- *Contextual resource discovery*: The dynamic state of the context requires the dynamic location and association of resources relevant to the user's context.
- *Context augmentation*: The user's context is enriched by digital data, which is presented accordingly to the user.

4.4 Aspects of computational distribution

Even though approaches featuring a centralized context management component are generally realizable, almost all of the current research emphasizes either distributed systems or hybrid approaches, where centralized component act as a subordinate system supplement. This fact implies, that software providing context-awareness is deployed as middleware on the nodes in the distributed system.

Since context-aware computing emphasizes the use of mobile devices, certain *hardware restrictions* must be considered [22]. Those especially encompass shortcomings in power supply, communication bandwidth, processing capabilities and storage. In addition, we face significant limitations in providing a user interface due to the lack of space.

Another interesting aspect of a decentralized system enables us to post the definition of *context spaces* [19], which has peripheral similarity to the context boundary aspect, introduced in section 3.3 [15]. A context space is set of nodes, that commonly share information. That implies a bounded locality of context allowing a distinction between local and global context aware applications: the local ones only access the contextual information in their context space, global context-awareness denotes the utilization of the global context consisting of all subordinate contexts in the respective context spaces.

4.4.1 Modularity

Since context-aware middleware needs to be deployed on a large number of devices, additional requirements regarding scalability and maintainability need to be considered. As stated before, the most significant architectural cut consists of separating the context management system from the context-aware application [3]. This enables the deployment of individual applications on the same context management system.

However, the context management system itself may have be applied to different use cases - basically requiring different context management systems. Hence, unitizing the context management system seems reasonable [20]. A component used by all context management system may be used as a generic component on all devices. This usually includes the acquisition and provision of contextual information. Specific context-processing components can then be deployed on top of the generic ones. They usually implement the mechanisms how context is individually processed (update, storage, context API).

4.5 Heterogeneity Aspects

As on context models, heterogeneity has an impact on middleware design as well [12]:

- *Hardware heterogeneity*: Servers, workstations and mobile devices in a context-aware system have very different characteristics and capabilities, especially concerning platforms, network technologies, computational power. The challenge in middleware design is to make it deployable on all of those device types.
- *Software heterogeneity*: The software running on the devices may differ highly, too. This encompasses the presence of different operating systems and applications.
- *Architectural heterogeneity*: The communication between various nodes may not be uniformly defined neither. This heterogeneity aspect comprises of the existence of different network architectures and communication protocols.

5 Conclusion

In this paper we have evaluated the current spectrum of research on context-aware computing and we have identified common characteristics among the diverse realization approaches. In this section, we summarize the results and conclude this paper by deriving a general architecture for context-aware systems.

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

Concluding from our analysis of implemented architectures we can derive a general conceptual architecture for

a context management system providing contextual information for context-aware applications. Figure 1 visualizes this concept employing all of the relevant components discussed earlier in section 4.1.

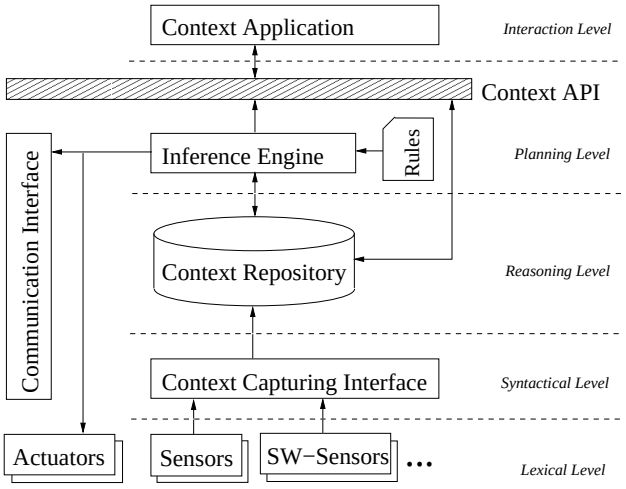

Figure 1: General Architecture

Additionally, the figure shows the architecture layered in levels of abstraction, as introduced in section 4.2. Each component has been assigned to the corresponding layer.

The workflow of recognizing, updating and utilizing context is strongly inspired by the procedure described in section 4.3. *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, and it represents the architectural cut between context management and context utilization as we have argued earlier and as it has been explicitly demanded by Christopoulou et al.[3]. 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.

At this point, we can distinguish 3 types of contextual updates:

- *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.

A question remaining after the discussions presented here is the utilization aspect of our work. Context-awareness is applicable in many utilization domains. On the one hand, the utilization of context is very suitable in distributed systems due to their dynamic behavior making context to change very quickly. This aspect makes mobile service development one the prime targets of our work. On the other hand, another interesting application domain is the development of internet services. Keeping this in mind, hybrid approaches implementing web-based mobile services are an important target group, too [1, 23].

An important aspect, which has yet been handled peripherally, is the presence of heterogeneity in several conceptualization domains. The analysis of this issue is the next step on our road map to progress in the project presented in this paper. It will allow us to refine our conceptualization under the aspect of the influence of heterogeneity factors.

The subsequent steps will aim at creating a prototype, which will basically implement the ideas presented in this paper.

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

[3] 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* [14], pages 265–270.

[4] 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.

[5] Mehdi Khouja, Carlos Juiz, Isaac Lera, Ramon Puigjaner, and Farouk Kamoun. An ontology-based model for a context-aware service oriented architecture. In *SERP 07: Proceedings of the 2007 International Conference on Software Engineering Research and Practice*, 2007.

[6] 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.

[7] Standard generalized markup language (iso 8879:1986). http://www.iso.org, October 2007.

[8] Composite capabilities / preferences profile. http://www.w3.org/Mobile/CCPP/, October 2007.

[9] User agent profile. http://www.wapforum.org, October 2007.

[10] Unified modeling language. http://www.uml.org, October 2007.

[11] Object role modeling. http://www.orm.net, October 2007.

[12] 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.

[13] Mohammad Rezwanul Huq, Nguyen Thi Thanh Tuyen, Young-Koo Lee, Byeong-Soo Jeong, and Sungyoung Lee. Modeling an ontology for managing contexts in smart meeting space. In *SWWS '07: Proceedings of the 2007 International Conference on Semantic Web and Web Services*, 2007.

[14] Eleni Christopoulou and Achilles Kameas. Gas ontology: An ontology for collaboration among ubiquitous computing devices. In *International Journal of Human-Computer Studies* [3], pages 664–685.

[15] Gruia-Catalin Roman, Christine Julien, and Qingfend Huang. Network abstractions for context-aware mobile computing. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 363–373, New York, NY, USA, 2002. ACM Press.

[16] Wieland Schwinger. Exploring model engineering techniques for the integration of heterogeneous context models. In *Proceedings of the 5th International Conference in Computing and Multimedia (MoMM2007)*, pages 65–76, 2007.

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

[18] Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Springer, Berlin (ISBN 978-0387952598), 2002.

[19] 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.

[20] Stephen S. Yau, Fariaz Karim, Yu Wang, Bin Wang, and Sandeep K. S. Gupta. Reconfigurable context-sensitive middleware for pervasive computing. *IEEE Pervasive Computing*, 1(3):33–40, 2002.

[21] 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.

[22] Roy Want and Trevor Pering. System challenges for ubiquitous & pervasive computing. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 9–14, 2005.

[23] 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. sterrecihische Computergesellschaft, 2007.