Designing, Developing, and Evaluating Context-Aware Systems*

Yoosoo Oh[1], Albrecht Schmidt[2], and Woontack Woo[1]
[1] *GIST U-VR Lab.*
Gwangju 500-712, S.Korea
{yoh, wwoo}@gist.ac.kr
[2] *Fraunhofer IAIS and b-it, University of Bonn, Schloss Birlinghoven, S.t Augustin, Germany*
albrecht.schmidt@iais.fraunhofer.de

Abstract

Context and context-awareness have been central issues in ubiquitous computing research for the last decade. Advances with regard to context acquisition and activity recognition allow interesting small scale applications. However in larger systems including many sensors and actuators and spanning multiple administrative domains are still remain as unsolved central issues. Particularly, in the areas of reasoning and context-fusion there are many open questions. In this paper we motivate large scale context-aware systems that have support for the full life cycle using a prototype that was implemented and tested. We then describe in detail the underlying architecture which supports dynamic context-aware systems and includes mechanisms for context fusion and reasoning. Also, we propose a new approach for evaluating context-aware systems. The approach is an adapted expert evaluation; well known in the user interface domain, but using a carefully selected set of heuristics specifically targeted at context-aware systems.

1. Introduction

Context-aware systems have been a research focus in ubiquitous computing since its very beginning [12]. Many systems have been developed in the research community using various sensors and acting on different contexts. However outside the research laboratories, context-aware systems have had little impact, with the exception of location aware applications. Location-aware applications (e.g. in car navigation) and location based services prove the usefulness of the overall concept as the first real commercial context-aware applications. Nevertheless, many issues with regard to more generic context-aware systems beyond laboratory scale are still unsolved. In particular the life-cycle support of context-aware systems (simulation, installation, debugging, and maintenance, adding new entities, removing old entities, and upgrading system components) has received little attention. These issues need to be addressed on an architectural level to provide generic solutions. In this paper we will introduce a new and open architecture for context-aware systems that provides mechanisms to solve these problems in parts.

Advances in context-aware computing have been published in the area of acquiring activity and context information using various sensors [2], [15]. This research indicates that for given situations it is possible to create sensors and recognize subsystems that provide context information to a larger system. Looking at the development cycle, it is of interest to support the simulation of context recognition. As indicated in recent work on prototyping [3], having manually simulated sensors can ease the early phases of development of context-aware systems. Similarly improvement in sensors and recognition algorithms are common and hence a system has to provide for changes on this level.

When designing and developing context-aware systems and architectures, one central issue is to show that the presented solution is appropriate and that it fulfills the criteria set. In the literature, several approaches can be found that evaluate the suitability of a solution. Often the argument is made that the architecture was successfully applied to one or more projects, e.g. [1]. In our view, this is not necessarily a good measure of the quality as it is usually not clear how much a specific solution relies on the presented architecture and where the clear benefits in the development and deployment were. Another common approach is that researchers compare their approach to

* This work was supported by Seondo project of MIC, Korea.

0-7695-2777-9/07 $25.00 © 2007 IEEE

a set of metrics they have chosen and potentially to similar systems published. This has the problem that there is always a bias, as the persons comparing the systems have much more insight in their own solution than in others. In this paper, we propose a novel architecture for context-aware computing systems and a new approach for evaluating context-aware system architectures. We use a modified heuristic evaluation process, which is well known in the domain of user interfaces, as means for evaluation of our approach.

The paper is structured as follows. In section 2, we present a prototype of a large scale context-aware system. In section 3, Context Integrator, the underlying context architecture is described. The evaluation of the architecture for context-aware systems is discussed in section 4. We report our findings of this case study and provide lessons learned for using heuristic expert evaluation for architectures.

2. Prototype of a Large Scale Context-Aware System

To explore issues related to large scale context-aware systems, the prototypical system was implemented and some parts are shown in Figure 1. The prototype is a context-aware building including a number of real and simulated sensors as well as a number of real and simulated actuators. We focused in the implementation on three different sensors: a wearable activity sensor, environment based status sensors, and an identity, PIM and interaction sensor on a mobile device. These sensors were implemented as real physical functional components as well as in simulated components. Having simulation as basic concept allows development before a real environment can exist. This is in particular interesting when developing context aware systems for new environments that do not exist. Since the context-aware software can be developed for where the software is required before the environment is ready. For example, while a new building is physically built, the software can already be developed and tested in simulation so that when the building is finished the tested software can be put to use.

The real input components have been implemented as typical representative of sensors we expect in a large scale context-aware system. The wearable activity sensor (ParticleExtractor) detects 3 axis acceleration data for a user's activity and posture, similar to the work described in [2]. Additionally it senses environmental data, such as sound, force, temperature, ambient light. As environment based status sensor, we implemented a component that runs on a PC and uses the login and Skype states as input (UserStatusExtractor). It show a typical sensor where basic information in the system is detected (here the user's state) and meta information is added (in this case the location and id of the PC) and handed onto the context-aware system for further processing and context recognition. An application on the mobile phone acts as an identity sensor, personal information management sensor and explicit control devices. It is assumed that the user manages her personal data (profile, schedule) on this mobile device. The sensor information is communicated via Bluetooth when a new environment is encountered; the level of detail (identity only or the whole schedule) is specified in the application. Providing this information to the system allows implicit interaction [9] with the environment. Additionally, the mobile phone application offers explicit controls for building environment (light, heater, and door).

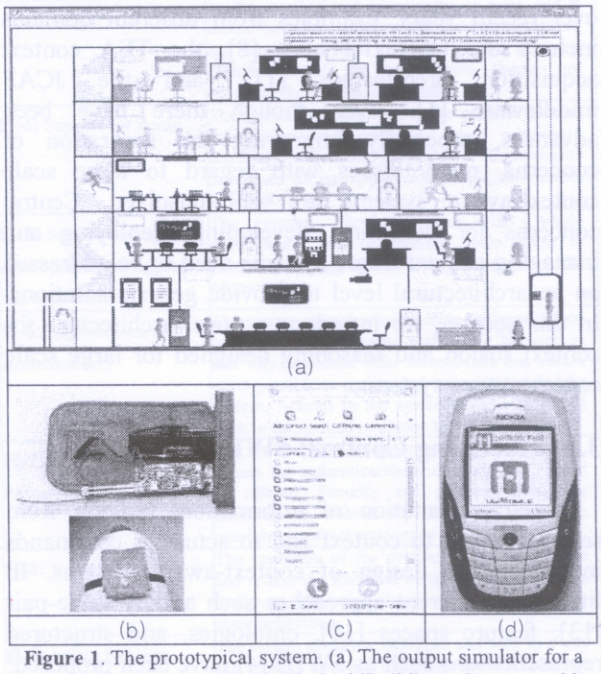

Figure 1. The prototypical system. (a) The output simulator for a large building implemented in Flash(ubiBuilding), (b) A wearable device that acts as human activity sensor and environmental sensor, (c) An extended Skype messenger which acts as a status sensor, (d) A mobile phone running the ubiMobile application.

The real output component can be any component in a real building system which can be controlled electronically, e.g. door locks, heating, shutters or lightning. In the prototype these are not implemented yet. For each real sensor we implemented a simulated counterpart that offers the same interfaces as the real sensor. The only difference is that the values are not sensed but selected in graphical user interface (GUI). These virtual sensors create context by clicking buttons

instead of converting context from data detected in real sensors.

The simulated output component is ubiBuilding Simulator which is a building simulation system implemented in Macromedia Flash, shown in Figure 1 (a). It simulates a full scale building which comprises 5 floors of 19 rooms with in total of 41 actuators (9 door-locks, 19 lights, and 13 heaters).

The central process of the system is context fusion module and computation engine. These central components are described in the next section in more detail.

3. A new architecture for context integration

Different architectures and middleware systems to support context-aware systems have been developed over recent years; examples from different domains include the Context-Toolkit [8], the TEA context acquisition architecture [11], and the JCAF middleware [1]. Even though there have been advances, especially improving the separation of concerns, many issues with regard to large scale context-aware systems are still unsolved. Central concerns for designing, developing, deploying and managing context-aware systems need to be addressed on an architectural level to provide generic solutions. In this section, we introduce a novel architecture for context fusion and reasoning designed for large scale context-aware systems.

3.1. Describing Context: 5W1H

The representation of information, ranging from sensor reading to context and to actuators commands influences the design of context-aware systems. In literature, different approaches such as key-value-pair [13], feature spaces [10], ontologies, and structured representations such as 5W1H [4] have been proposed. The architecture described in this section uses the 5W1H context representation as its basis.

The 5W1H Context is the structured format, where context is expressed as its components with regard to Who, What, Where, When, How, and Why. We used this representation to provide a user centric view of context which is appropriate for the anticipated systems we want to support with the architecture. Using the 6 questions and structuring information gathered by sensors accordingly, allow us to accurately represent the user's situation as it can be observed by sensors. In addition this context representation can reduce the management and provision of context according to individual services.

3.2. Context Integrator Architecture

Context fusion and reasoning are the central functionality provided by the Context Integrator, which is a novel architecture for context-aware systems. An architectural overview is shown in Figure 2.

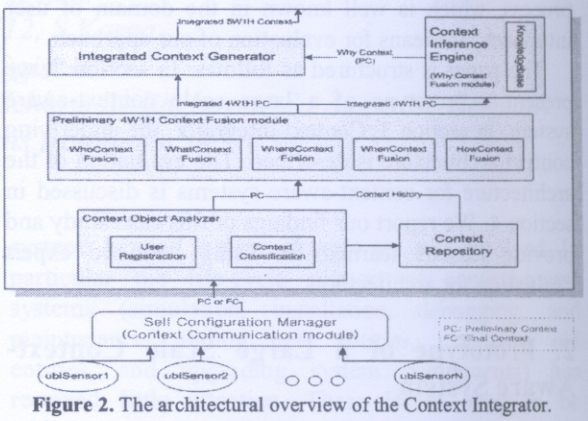

Figure 2. The architectural overview of the Context Integrator.

The Context Object Analyzer collects contexts (preliminary or final alike) periodically from various kinds of sensors which are placed in the same active area. The Context Repository stores and manages the history of the integrated contexts. The Preliminary Context Fusion module integrates the inputted Preliminary Contexts as an integrated 4W1H context according to characteristics of each sub-context of 4W1H (Who, What, Where, When, and How). Preliminary Context (PC) expresses the feature information and sensor description. Integrated Context (IC) forms the complete 5W1H context generated by Context Integrator which additionally includes the inferred intention using the context inference engine. The Context Inference Engine reasons the 'Why' component of the context (intention) by using the result of the Preliminary Context Fusion module. It contains a knowledgebase which consists of facts and rules that describe the behavior of the system. Finally, the Integrated Context Generator combines all components into a 5W1H Integrated Context. This contains information, such as user's identity, location, activities, behavior, patterns, and intention.

The Context Integrator is designed to provide information fusion on different levels and support fusion of sensor data from various sources. This results in a system that accomplishes symbolic context fusion. Fusion on raw data level is considered as a part of the sensor internals. By classifying and fusing according to the user ('Who' dimension), a user-centric context integration is supported.

The Context Integrator has built in mechanisms for reuse of contexts in particular in the context fusion step. The reuse occurs in the inference chain which reuses the reasoned result of the previous step. With an increasing number of inference steps, higher-level context can be created. This is the mechanism used to detect intention ('Why' dimension). Additionally the reuse is achieved by updating and using recorded contexts in the context history which are held in the Context Repository.

The generic behavior and functionality of a specific context-aware system based on the Context Integrator is specified by rules. Changes and reconfiguration of the system by a developer or maintainer are done by adding, deleting, and modifying rules. For simplicity, these rules are specified in plain text and reconfiguration just requires a text editor. Hence changes to the overall system behavior do not require changes in any source code in Context Integrator. The Context Inference Engine is implemented on top of JESS (Java Expert System Shell).

Support for authentication can be added to the Context Integrator. It can be specified that contexts are only taken into account for certified and authenticated users. To maintain privacy, the user has control of what information can be gathered by the sensors.

3.3. Explanation and Accountability

To make it feasible for developers and operators to understand how and why a system is performing certain actions, a component for explanation was added. In the prototype this was also implemented in Flash to allow easy changes and provide more options for visualization. The example shows the intention output message which describes the user's current intention with her current activity as a point of view of an outsider. For easy debugging to developers, the weighting factor to the current output is calculated and the previous information (activity, location, number of persons) is revealed, as shown in Figure 3.

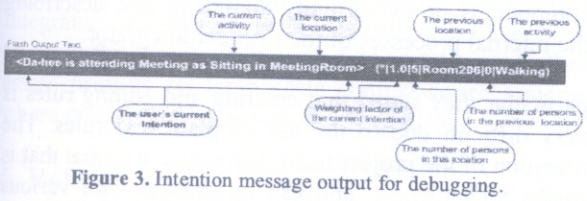

Figure 3. Intention message output for debugging.

4. Heuristic Evaluation of Context-Architectures

Evaluation of architectures and middleware is a topic that is often discussed and regarded as very difficult [14]. Especially in application driven research, finding appropriate means for assessment of quality often leads to disagreement between experts whether or not a system is appropriately evaluated. Our approach is an adapted expert evaluation that is successfully used in the domain of user interfaces evaluation [6], [7]. It has been extended to ambient displays, a special type of output user interfaces, as presented in [5].

Table 1. Eleven heuristics.

H-1: Separation of Concerns	Sensors, services, and applications should be separated. Changes in one of these components should be independent of the others.
H-2: Flexibility and Openness	The approach should be flexible and open to allow integration of (virtual) components and change of components. The system should be open so that other components (not related to the original system) can be integrated.
H-3: Scalability	The complexity of the system should dependent on the number and complexity of rules and it should be independent of the number of sensors and actuators as well as of the number of users. Distribution of the components should not influence the overall performance.
H-4: Support for Reuse	The approach should support the reuse of context information and contextual data previously acquired (context-history). Components developed for one application should be reusable in others.
H-5: Support during development, debugging and deployment	The system should support the whole development and deployment cycle. Here in particular debugging and error tracing should be supported.
H-6: Explanations and Accountability	The system should provide clear explanations on how the system behaved and why it behaved in this way. It should offer information that makes users possible to understand actions taken by the system. Actions in the system should be traced back to the context or person that initiated them.
H-7: Security and Privacy	The system should support security and privacy. Means for authentication and encryption should be offered. Security and privacy should be provided at architecture level. It should be possible to restrict services to certain users.
H-8: Reliability	The operation of the system should be reliable and reflect the intention of the users and system designers. The same situation should result in the same behavior.
H-9: Match between context-aware system and the real world	The system should be designed and built in a way that there is always a match between the context-aware system (the state observed by the system) and the real world. Changes in the real world should have an immediate effect in the context-aware system.
H-10: Manual override	Users should have means for manual overriding decisions taken by the system based on context information.
H-11: Reconfiguration and Management	The system should be easy to configure, re-configure and manage when deployed. It should be possible to easily change and adopt the system behavior during runtime.

4.1. Heuristic Evaluation of Context-Aware System Architectures

The basic idea of an expert evaluation using heuristics is to have a small number of experts (4-6) assessing a system based on a set of heuristics, as shown in Table 1. The experts are required to be knowledgeable in the field of the domain (in our case with context-aware systems). The experts performing the evaluation should not have an in-depth knowledge of the system. This excludes all people working on the project as expert evaluators. For each heuristic they assess the system and look whether or not the heuristic is violated. When they find issues that are violated they rate the severity of the problem on a scale of 1 (minor problem) to 5 (imperative to fix). During the design and development process, however the heuristics can provide very valuable guidance for the project team.

There are significant differences in evaluating user interfaces and system architecture or middleware. The difference is that the functionality of the user interface is visible and must be usable on the basis as presented. In contrast system architecture and applications based on it are less visible and hence not clearly assessable by observation. In our approach for a heuristic evaluation we therefore suggest that the reviewers are provided with the working system, the documentation and applications for the system. They then get an introduction to the system. Reviewers familiarized themselves with the heuristics and the system. The project team was available for questions.

4.2. Case Study: Evaluation of a novel Context-Architecture

To improve and evaluate the above described context-architecture, we performed a heuristic evaluation. All expert evaluators had experience with designing and developing context-aware systems. 4 had a master degree in computer science or a related field and one held a PhD. None of the experts was involved in the design or development of the system. The experts spend on average 1.2 hours for the evaluation. During this time they could communicated with a member of the project team if they needed clarification. After completing the review they provided the detailed description of heuristics violated and rated the severity of each violation individually. Additionally the evaluators were asked to give general comments about the system that was evaluated and about the evaluation process. In the following paragraphs we discuss the 4 most criticized heuristics. In Figure 4, the accumulated severity ratings for all heuristics are depicted.

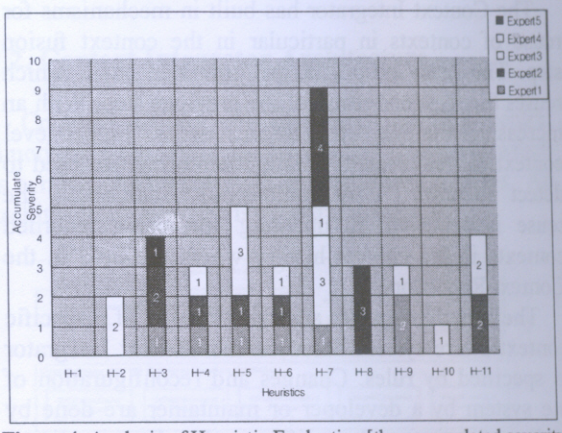

Figure 4. Analysis of Heuristic Evaluation [the accumulated severity ratings].

The heuristic H-7 was seen violated by 4 out of the 5 experts; where 2 saw it as significant and 2 as minor problem. Security and privacy have been an issue in the project team but never became prominent within the architecture. From a viewpoint of the architecture these are add-on feature, however given the criticism from the evaluation this is a point to be reconsidered in the next iteration of the design.

The second serious problem was related to H-5 and especially to issues related to debugging. This was very surprising for the project team, as supporting the life cycle was one of the central aims when designing the architecture. We assumed that providing explanations of why events were generated and why actions were taken, as depicted in Figure 3, would be sufficient to support debugging. However the experts saw this only as a starting point and suggested more advanced functionality to be added (e.g. history trace).

Also the problem addressed by 3 experts but with lower severity was H-3. Without an in-depth knowledge of the context-processing (rule engine) it is difficult to see what effects a large number of sensors and actuators will have. This indicates that for a further version, scalability needs to be ensured by figures either based on simulation with a large number of components or by formally analyzing and describing the internal processes of the Context Integrator.

2 experts indicated that H-11 is violated, in particular they stated that creating and editing rules is very difficult due to the use of plain text rules. The intention of the project team was to use a format that is easily exchangeable and can be created with various editors (for further work a GUI-based editors for rules is included). The criticism indicates that the priority for developing such a GUI component should be increased.

In the general comments, one expert reviewer addressed the use of templates in combination with

rules to improve reconfiguration. Additionally he also raised issues for the need of policies for dealing with context history information. In the comments that relate to the evaluation process, the reviewers found it in parts difficult to relate them to the concrete issues in the architecture they evaluated. This experience is similar to doing heuristic evaluation in user interfaces, here often the problem of addressing the concrete issues based on abstract heuristics arises, especially if people have not done expert evaluations before. The selection of heuristics was discussed in the comments; however it appeared that the heuristics stated were useful, the evaluators suggested that there may be more heuristics than need to be assessed. This is a trade-off for this type of evaluation. One comment indicated that the heuristics may be viewed and evaluated from different viewpoints, e.g. from a user's perspective and from a developer's perspective. This appears as an interesting issue for following evaluations.

5. Conclusion

In this paper we have introduced Context Integrator, a novel architecture for context-aware systems. The proposed architecture has been used to develop prototypical systems. Its central component is a rule based system that creates a system behavior based on contexts detected. Using rules as the basic means are a flexible way to change and manipulate the system behavior during all phases of the life cycle of a context-aware system. The central advantage for large scale context-aware systems is that a change in behavior can be introduced at run time by adding, changing or deleting rules.

We presented in a case study our experience with using a modified heuristic expert evaluation for context-aware system. Based on 11 heuristics which we have proposed, we experience very valuable feedback for the project and a critical evaluation providing new insights that would have not been gained within the project team alone.

Currently we are further developing the Context Integrator architecture to fix issues that have been found in the expert evaluation. Additionally we further assess projects on large scale context-aware systems where the developed system can be applied.

6. References

[1] Jakob E. Bardram, "The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications," In Proc. of Pervasive 2005, LNCS, Springer-Verlag, 2005, pp. 98–116.

[2] L. Bao and S. S. Intille, "Activity recognition from user-annotated acceleration data," In Proc. of Pervasive 2004, LNCS 3001, Springer-Verlag, 2004, pp. 1-17.

[3] Steven Dow, Blair MacIntyre, Jaemin Lee, Christopher Oezbek, Jay David Bolter, Maribeth Gandy, "Wizard of Oz Support throughout an Iterative Design Process," IEEE Pervasive Computing, vol. 4, no. 4, 2005, pp. 18-26.

[4] S.Jang, W.Woo, "ubi-UCAM: A Unified Context-Aware Application Model," In Proc. of Context2005, LNAI, vol. 2680, 2003, pp. 178-189.

[5] Jennifer Mankoff, Anind Dey, Gary Hsieh, Julie Kientz, Scott Lederer, and Morgan Ames, "Heuristic evaluation of ambient displays," In Proc. of ACM Conf. on Human Factors and Computing Systems (CHI 2003), 2003, pp. 169-176.

[6] J. Nielsen, Enhancing the explanatory power of usability heuristics, In Pro. of ACM CHI 1994, 1994, pp. 152-158.

[7] J. Nielsen, "Heuristic Evaluation," http://www.useit.com/papers/heuristic/, 2006

[8] D. Salber, A.K. Dey and G.D. Abowd, "The Context Toolkit: Aiding the Development of Context-Aware Applications," In Proc. of Human Factors in Computing Systems: CHI 99, ACM Press, 1999, pp. 434-441.

[9] A. Schmidt, "Implicit Human Computer Interaction Through Context," Personal Technologies, 4(2&3), Springer-Verlag, 2000, pp. 191-199.

[10] A. Schmidt, M. Beigl, and H.-W. Gellersen, "There is more to context than location," Computers and Graphics, vol. 23, no. 6, 1999, pp. 893-901.

[11] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven, and W. Van de Velde, "Advanced interaction in context," In Proc. of First International Symposium on Handheld and Ubiquitous Computing (HUC99), vol. 1707, LNCS, Springer-Verlag, 1999, pp. 89-101.

[12] B. Schilit, N. Adams, and R. Want, "Context -Aware Computing Applications," In Proc. of Workshop Mobile Computer Systems and Applications, IEEE CS Press, 1994, pp. 85-90.

[13] B. Schilit, Marvin M. Theimer, and Brent B. Welch, "Customizing mobile applications," In Proc. of USENIX Mobile & Location-Independent Computing Symposium, 1993, pp. 129-138.

[14] R. Sharp, K. Rehman, "What Makes Good Application-Led Research?," IEEE Pervasive Computing, vol. 4, no. 3, 2005, pp. 80-82.

[15] K. Van Laerhoven, A. Schmidt and H.-W. Gellersen, "Multi-Sensor Context-Aware Clothing," In Proc. of the 6th Int. Symposium on Wearable Computers, ISWC 2002, IEEE Press, 2002, pp. 49-57.