CAKE – Distributed Environments for Context-Aware Systems

Jörg Cassens[1], Felix Schmitt[2], and Michael Herczeg[2]

[1] Institute for Mathematics and Applied Informatics,
University of Hildesheim, Samelsonplatz 1, 31141 Hildesheim, Germany
`cassens@cs.uni-hildesheim.de`

[2] Institute for Multimedia and Interactive Systems,
University of Lübeck, Ratzeburger Allee 160, 23562 Lübeck, Germany
`{schmitt,herczeg}@imis.uni-luebeck.de`

Abstract. In this paper, we introduce the distributed Context Awareness and Knowledge Environment CAKE. The design objectives for CAKE were to develop a system that is flexible enough to be used in different application domains, that supports re-use of components with the help of a well-defined plugin-system and application programming interface and that caters for privacy concerns by giving users access to personal context aware environments that share information selectively with other users' context aware environments. We describe related work on context middleware and the niche CAKE is targeting. We also argue for taking privacy concerns into account and outline how our framework addresses such issues. The concepts behind CAKE are introduced, and we describe how reasoning engines based on different paradigms can be put to work together in our framework. A first take on end-user programming is outlined and a prototypical implementation of the system presented.

Keywords: Context Awareness, Context Middleware, Distributed Systems.

1 Introduction

We have previously developed a framework for ambient intelligent systems [16,17], but its architecture had two main shortcomings. The first one was a lack of flexibility and versatility with regard to how the system interacts with sensors and actuators, and the second one was a lack of ability to deal with user concerns about privacy. These shortcomings are a result of the genesis of the project: it started as a system to support coordination and communication in small teams of knowledge workers.

To target the original application domain, we developed a centralized architecture that supported the whole team, and we introduced specific protocols for communicating with sensors and actuators based on the Extensible Messaging and Presence Protocol (XMPP)[1]. From a technical point of view, this meant that every sensor and actuator had to either implement the whole stack up to the XMPP-layer, including

[1] `http://xmpp.org/`

having its own XMPP-ID (or JID, for Jabber ID), or that they had to be connected to an aggregator driving the specific sensor or actuator. This was not a problem for teams of knowledge workers, since we could assume that the user's PC would be used to connect "dumb" devices with the central hub, but it placed an increasing burden on the development of novel input and output devices for different application domains.

The centralized hub and spoke architecture introduced another set of problems. While it was not unreasonable for small teams where the members trusted each other, it became increasingly clear both from our own experience and the literature [3,12,14] that users might not accept to share raw sensor data with a centralized architecture where they had no control on how this data was to be used.

Therefore, it was decided to design a new framework building upon the lessons learned with the existing system, eliminating the weaknesses identified while retaining its strengths. The latter part is also the reason for not abandoning our own framework completely in favor of one of the existing middleware solutions.

2 Related Work

The goal of the AmbieSense project [11] is to provide an "ambient landscape" where personal, mobile computers connect to so-called context tags in the environment in order to access context parameters. The devices can also make use of net-based information services. Communication and coordination of different personal devices is not the main focus.

The ASTRA project [15] realizes a pervasive awareness system where personal pervasive systems are connected to deliver information about the state of users to each other. ASTRA provides means for end-user development in the form of rule sets that define what information is shared and how one's own environment reacts to changes in other users' environments. Privacy is a core aspect in ASTRA, but local reasoning capabilities are limited.

UbiCollab [5] is a toolkit that supports collaboration in ubiquitous environments, e.g. using mobile phones. It is a mature platform for developing mobile and ubiquitous applications with a strong focus on CSCW-aspects.

The Integ Smart Home System [13] uses a wireless, ad-hoc sensor network to allow users to control their homes over the internet. The application domain and the solutions to connect sensors and actuators via abstracted interfaces are very relevant for our own work. However, the collaboration support is not very elaborated, and underlying reasoning mechanisms seem to be restricted to a rule-based system.

The POSTECH U-Health Smart Home project [9] targets ambient assisted living, another interesting application domain. U-Health also uses abstraction techniques to access different system types. When it comes to connecting different environments, the main focus is the ability to notify help personnel in case of emergencies. U-Health exhibits learning capabilities to adapt to its users.

Ambient Dynamix [1] is a framework for connecting mobile applications and websites with sensors and actuators in the real world. An interesting feature of Dynamix is the concept of a context firewall, which allows different applications to limit access to contextual information on a fine-grained level. The main focus lies on services for mobile phones, and not so much on context reasoning.

3 Concept and Vision

The analysis of the literature combined with our own lessons learned lead to the following goals, which became central to the development of the new system:

1. **Flexibility** with regard to **sensors** and **actuators**: it should be comparatively easy to connect new devices with the system, both for developers and users:
   (a) It should be easy for developers to write new software components, and
   (b) users should be able to easily and safely add such components by their means.
2. **Flexibility** with regard to the **reasoning engines** that can be used, in order to utilize different reasoning paradigms based on their suitability for the task at hand.
3. **Reusability** of sensors and actuators across different domains, where applicable.
4. A **decentralized** architecture that would allow every user to run his or her own context-aware environment to address concerns about sharing raw data.
5. Further addressing **privacy** concerns by allowing for fine-grained and coarse-grained control of who can access what information.
6. **Feature parity** with the existing system when it comes to sensors, actuators, reasoners and simulators.

4 Design and Architecture

In this section, we describe the basic design and architectural features of the new system called CAKE (Context Awareness and Knowledge Environment). It is a distributed system that allows every user to run his or her own CAKE instance and grant other instances only access to selected raw or processed data. In the domain of team coordination, for example, the user might give other team members access to limited information only, while providing personal friends with more information from the same CAKE instance [16].

CAKE is a modular system, based on a strict separation of concerns, where the different modules are only loosely coupled. The system is implemented predominantly in JAVA. CAKE encompasses four different modules, as described in the following.

Plugin-Management. This module allows adding and removing sensor and actuator plugins at runtime. It abstracts sensor data according to CAKE's knowledge model. A plugin provides connectivity for a specific sensor or actuator. The API demands a manifest describing the plugin, a description how to calibrate the attached device, the update rate and a unique ID plus version information. Plugins can be published in a repository to be searched for and downloaded by other CAKE instances. Calibration information can be used to calibrate sensors or actuators from inside the running CAKE instance. For security reasons, each of the plugins runs in its own sandbox.

Logic. The logic module represents the known state of the world and connects the different reasoning engines to work on and transform the information known. CAKE uses a whiteboard to assemble all the information the system has about the world and the state of its reasoners. The whiteboard has been inspired by multi-agent blackboards [4], but it does not support the whole functionality needed for multi-agent systems, like for example the control shell. Sensor values are written on the whiteboard, and the different reasoners can subscribe to changes of these values. Results of the reasoning process are again written onto the whiteboard. Those results can be aggregated to "virtual sensors" which again can serve as input to other reasoners, or the results can be used to change the state of actuators. Per default, information on the whiteboard is only available to the CAKE instance where it runs, but it can selectively be made accessible by other CAKE instances.

The world model used by the whiteboard is described in terms of an RDF graph, but the system is agnostic with regard to the reasoning paradigms used by the different reasoners. Attached reasoning engines can read the state of the whiteboard through the RDF graph or via JAVA methods. Updates can, for now, only be done via JAVA methods. The reasoners can add to the ontology that describes CAKE's world model by supplying additional RDF graphs. Reasoners that work on RDF representations can be added directly. Developers of reasoners using different paradigms have to provide a mapping of their internal representation and the corresponding RDF model. A myCBR-based[2] case-based reasoner is being integrated.

A simple reasoner based on production rules [7] is built into the system. Users can use this reasoner to develop their own production system, where they define how actuators should react based on different sensor parameters. This provides basic end-user development capabilities. However, making users define what basically are complex, hierarchical if-then rules has poor transparency for non-expert users. Therefore we are looking into other paradigms for end-user development as well.

Communication. The communication module provides a REST interface [6] for the graphical user interface and connects to other CAKE instances through XMPP. For the GUI part, it is possible to configure actuators or sensors, define new rules and add or modify users, groups and permissions. Users who are given access to local data are represented by their XMPP-ID (or JID). Specific XMPP-extensions to facilitate connecting different context-aware systems have been developed and make it possible to connect to CAKE instances from other types of systems as well.

GUI. The web-based GUI-component adds user-facing administrative capabilities such as user and group management, plugin management and the definition of production rules as a means of end-user programming of context reasoners. The idea is that the CAKE system can be deployed similarly to routers or modems: small appliances that come with the necessary connectivity and are set up by the end user.

[2] `http://mycbr-project.net/`

5 Conclusions and Further Work

Looking at how CAKE interfaces with its environment, the first goal of *flexibility* for developers with regard to *sensors* and *actuators* is supported by a plugin architecture that helps developers to write software components to access new sensors and actuators. Flexibility for end users is supported by the integrated web GUI. Here, users can not only perform administrative tasks like adding and removing sensors and add, change or remove users and groups who are allowed to access one's own CAKE instance, but also define basic production rules to perform end-user development.

In order to support the second goal of retaining *flexibility* with regard to *reasoning* about context, the integrated whiteboard architecture allows for different application-specific and general-purpose reasoners to access sensor information, change the state of actuators and put processed information back onto the whiteboard for use by other reasoners or sharing with other CAKE instances.

The third goal, *reusability,* is supported by the ability to make plugins available to others and install and configure new plugins at runtime. Reusability of reasoners is, for the time being, supported in a limited way. Reasoners can only be added or changed when the system is not running, and updates from the reasoners are limited to the JAVA object interface and cannot be done through manipulating the RDF graph.

CAKE instances can be run by individual users or groups, and the integrated networking components make it easy to connect several instances so that e.g. members of work teams can get access to their peers' interruptibility status. Authorization and authentication of different instances are handled by the underlying XMPP layer. This fulfills the fourth goal of designing a *distributed system.*

Privacy issues, concerning the fifth goal, are handled by each instance through user and group management, where different users or groups can get fine- or coarse-grained permissions to access information or raw data. However, this approach does not scale very well, and issues such as minimizing asymmetry in information flow [8] can only be addressed by introducing meta-reasoners that monitor the information flow. Further work in this direction, like adding the ability to add proxies for privacy [10], has to be conducted.

Finally, the sixth goal of *feature parity* with the existing system has not been reached yet. Several sensors and actuators from previous incarnations are not available, and not all reasoners have been ported. However, we hope that we will able to port those entities on a case by case basis, if the existing solutions prove worthwhile to keep. This is especially true of our own simulation environment [2].

Acknowledgments. The authors wish to thank their students for their contributions in designing and implementing CAKE: David Bouck-Standen, Tim Dubbels, Bjørn Eberhardt, Eva Jehle, Sandro Kock, Alexander Schulze and Daniel Wilken.

References

1. Carlson, D., Schrader, A.: Dynamix: An open plug-and-play context framework for Android. In: 3rd International Conference on the Internet of Things (IOT), pp. 151–158. IEEE, New York (2012)
2. Cassens, J., Schmitt, F., Mende, T., Herczeg, M.: CASi – A Generic Context Awareness Simulator for Ambient Systems. In: Paternò, F., de Ruyter, B., Markopoulos, P., Santoro, C., van Loenen, E., Luyten, K. (eds.) AmI 2012. LNCS, vol. 7683, pp. 421–426. Springer, Heidelberg (2012)
3. Consolvo, S., Smith, I.E., Matthews, T., LaMarca, A., Tabert, J., Powledge, P.: Location disclosure to social relations: why, when, & what people want to share. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 81–90. ACM, New York (2005)
4. Corkill, D.D.: Blackboard systems. AI Expert 6(9), 40–47 (1991)
5. Divitini, M., Farshchian, B.A., Samset, H.: UbiCollab: collaboration support for mobile users. In: Proceedings of the 2004 ACM Symposium on Applied Computing, pp. 1191–1195. ACM, New York (2004)
6. Fielding, R.T.: Architectural styles and the design of network-based software architectures. PhD Thesis, University of California, Irvine (2000)
7. Giarratano, J.C., Riley, G.D.: Expert Systems: Principles and Programming. Brooks/Cole Publishing Co., Pacific Grove (2005)
8. Jiang, X., Hong, J.I., Landay, J.A.: Approximate information flows: Socially-based modeling of privacy in ubiquitous computing. In: Borriello, G., Holmquist, L.E. (eds.) UbiComp 2002. LNCS, vol. 2498, pp. 176–193. Springer, Heidelberg (2002)
9. Kim, J., Choi, H., Wang, H., Agoulmine, N., Deerv, M.J., Hong, J.W.-K.: POSTECH's U-Health Smart Home for elderly monitoring and support. In: IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM), pp. 1–6. IEEE, New York (2010)
10. Kofod-Petersen, A., Cassens, J.: Proxies for Privacy in Ambient Systems. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications 1(4), 63–75 (2010)
11. Kofod-Petersen, A.: A Case-Based Approach to Realising Ambient Intelligence among Agents. PhD Thesis, Norwegian University of Sci. and Tech. (2007)
12. Lederer, S., Manko, J., Dey, A.K.: Who wants to know what when? Privacy preference determinants in ubiquitous computing. In: CHI 2003 Extended Abstracts on Human Factors in Computing Systems. ACM, New York (2003)
13. Mantoro, T., Ayu, M.A., Elnour, E.E.: Web-enabled smart home using wireless node infrastructure. In: Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia, pp. 72–79. ACM, New York (2011)
14. Nissenbaum, H.: Privacy in Context: Technology, Policy, and the Integrity of Social Life. Stanford University Press, Palo Alto (2009)
15. Romero, N., Markopoulos, P., van Baren, J., de Ruyter, B., Ijsselsteijn, W., Farshchian, B.: Connecting the family with awareness systems. Personal and Ubiquitous Computing 11(4), 299–312 (2006)
16. Ruge, L., Kindsmüller, M.C., Cassens, J., Herczeg, M.: How About a MATe for Awareness in Teams? In: Proceedings of Context 2011, pp. 58–69. Springer, Heidelberg (2011)
17. Schmitt, F., Cassens, J., Kindsmüller, M.C., Herczeg, M.: Mental Models of Ambient Systems: A Modular Research Framework. In: Beigl, M., Christiansen, H., Roth-Berghofer, T.R., Kofod-Petersen, A., Coventry, K.R., Schmidtke, H.R. (eds.) CONTEXT 2011. LNCS, vol. 6967, pp. 278–291. Springer, Heidelberg (2011)