

An Architecture to Enable Remote Context Reasoning

Michael Przybilski, Petteri Nurmi
Helsinki Institute for Information Technology (HIIT)
Basic Research Unit
University of Helsinki, Department of Computer Science
P.O. Box 64, FI-00014, Helsinki, Finland
{firstname.lastname}@cs.helsinki.fi

Abstract—A step that many context-aware systems have in common is that of context reasoning. A major problem this research is currently facing, is the absence of real-life data, which would enable the verification of the usefulness of the reasoning mechanisms for specific data-sets. In this paper we describe a server mechanism that enables the flexible collection and storage of context data from multiple users and sources. The server furthermore provides a notification service so that the collected data can be further used by different tools, such as different reasoning mechanisms. The system has been implemented using J2EE technologies and it has been tested with clients, which transfer context data to the server, using SOAP on top of HTTP.

Keywords—ubiquitous computing, software architecture, distributed AI, software infrastructures, adaptive systems.

I. INTRODUCTION

One of the emerging technologies in the field of computer science is context-aware computing. An important part of this research deals with the reasoning about the current context of a user, or another entity, in order to infer higher levels of abstraction of the context. These abstractions are typically used in order to make applications less dependent on specific data sources and sensing mechanisms.

In order to improve reasoning in context-aware applications, novel reasoning mechanisms (e.g. [1]) and architectures for context-aware systems (e.g. [2]–[5]) are being researched. One problem that the research is facing, however, is the absence of real life data. This data could be used to verify new

approaches without spending major effort on the implementation of surrounding applications and the means for gathering context data.

While learning and inference mechanisms can be evaluated with simulated data, there is a high probability that the approach fails on real-life data-sets. The development and implementation of the simulation mechanism itself often requires significant effort, but offer no assurance that the reasoning mechanism, verified with this simulation, can hold up to the kinds and amounts of context information the final system may be subjected to. Furthermore, combining many different context-sources and reasoning about their relationships provide by themselves a very good source for higher level context, which can only be simulated with great effort.

For these reasons a system that can gather different kinds of context data from many sources is very beneficial. Such a system should furthermore provide a consistent interface to all kinds of reasoning systems that could then be verified, using the collected data.

The remainder of this paper is organized as follows: Section II discusses the background for this work, while specific requirements are introduced in Section III. The architecture of the system is then presented in Section IV, whereas Section V presents details regarding the implementation of the system. Finally, Section VI concludes the paper and provides directions for future work.

II. BACKGROUND

Current systems that support the study of context-awareness are usually built with the specific purpose of the study in mind (see for instance [6]) and it is usually very difficult to extend those systems for other purposes, or to integrate them with existing applications. As also Winograd recognizes, so far only very few common architectures, or suggestions towards the architecture of context-aware systems (see for example [7]) have been developed [8].

Different approaches have been taken to provide a common architecture for context-aware applications and towards the integration of context-reasoning mechanisms. According to Moran and Dourish [6], current research focuses on either different versions of a blackboard-based approach or on widget-based approaches, which are in most cases implemented in the form of middleware and its services, or in the form of application frameworks. Another possibility that Moran and Dourish describe is implementation using interacting agents, which are distributed over a network [4].

Examples of middleware approaches include, the Reconfigurable Context-Sensitive Middleware (RCSM) [9], and the CORTEX middleware [10]. For our purposes the CORTEX approach is more interesting as it introduces special entities, called *sentient objects*, which are responsible for receiving, processing, and providing context-related information. Sentient objects are defined as autonomous objects that are able to sense their environment and act accordingly [10]. The advantage of this approach is the possibility to re-organize them, for instance depending on their primary task.

Also some application frameworks that support context-awareness have been proposed. For example, [11] describes the implementation of a framework that supports management of context information on mobile terminals. The structure of this framework is centered around the blackboard paradigm for communication, which is handled by a context manager. Most components that use this framework, including the applications, act as clients for the context management system and reside on the device itself. Other services can run not only on the device itself, but potentially also in a distributed environment.

Another framework approach is The Context Toolkit, which separates acquisition and presentation of context information from the application that requires it, by using so-called widgets [12]. The focus of this work lays in the automatic inference of higher level context information from lower-level sensory data.

Another system, developed within the Hydrogen project [13], describes a three-layered architecture, designed to overcome problems of mobile, context-aware systems. The framework consists of the adaptor, the management and the application layer. For the communication between the different layers, the described framework utilizes an XML-based protocol. The aim of the framework is the provision of an architecture that is lightweight, extensible, robust, and which also enables the possibility of adding further meta-information to the system

III. REQUIREMENTS

Although the systems described in the previous section have been developed to facilitate development of context aware systems and applications, they are unnecessarily complex for the purposes of gathering context data and verifying different reasoning techniques. In this section we identify functional requirements for a system that is suited for the above mentioned purposes. The architecture itself is discussed more thoroughly in Section IV.

Although the resulting system should be kept simple, it should be able to gather data from many different devices and users. Especially mobile devices have inherently access to huge amounts of data and are thus very suitable for collecting contextual information. Due to the limited resources of a typical wireless information device, it is however often necessary to upload the gathered information to a system, which has more resources available. Because of the potentially large amount of end users, the resulting system also needs to be highly scalable.

The system should also provide an interface for users to manage context information, gathered from their devices. Thus, it should be possible for a user to browse the data (s)he has provided and delete information that was provided by mistake or inadvertently.

In order to keep the resulting system as flexible as possible, the automatic adaptation to new types of context data and new context formats is needed. Since the system has to also store the context information, the specification of the context data format can furthermore be used to adapt the data storage mechanisms.

In order to verify the behavior of different reasoning mechanism on realistic data-sets, a mechanism is needed that enables the provisioning of gathered context data to different third party tools and mechanisms, and in many different formats. This mechanism should furthermore be able to provide the context data in a timely manner and in such a way that an external system is able to receive incoming context data and recognize and react to context changes. Thus, an event model is needed that can push notifications of new context information to external systems.

Finally, to ensure security of the system, it is necessary to authenticate users and their devices. However, since the security aspect is not the primary concern of the system, it is merely necessary to provide an interface towards external systems that implement the authentication of users and their devices, based on the provided credentials.

IV. ARCHITECTURE

The requirements discussed in the previous section have lead to the architecture shown in Figure 1. The provided interfaces are the *Mobile* interface, which allows mobile devices to supply context data, and the *Browser* interface, which enables the management of contextual information for the end-users and the administration of the system. What is required from external components is the *Authentication* of users and devices. The *Reasoning* interface enables the connection of different reasoning mechanisms, which can then be validated with the provided data. The sub-components of the system are discussed in more detail in the following subsections.

A. Data Collection

The first step of the data collection process is implemented by the *Context Listener* component, which receives incoming messages. Using external authentication mechanisms, the *Context Listener*

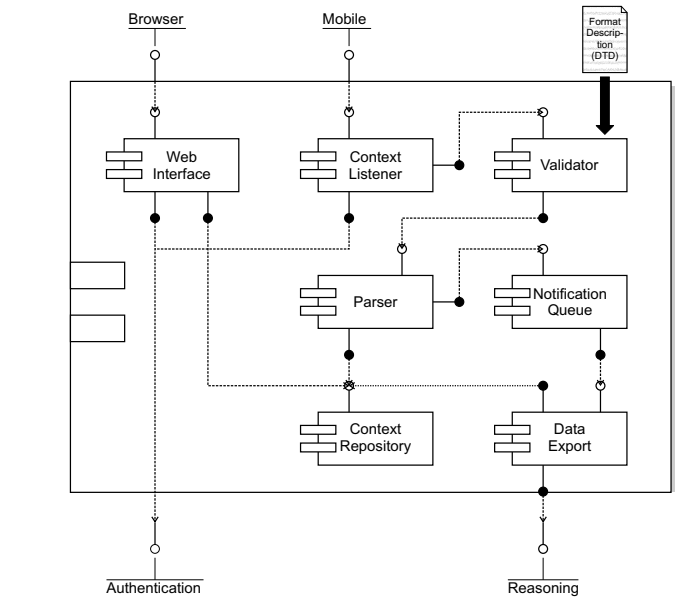


Fig. 1. Components of the Context Management Server

verifies the transmitting device. If the credentials of the device are valid, the context data is passed further to the *Validator*. On the other hand, if the device is not permitted to use the system, according error information is provided to the device and further processing of the message does not take place. For logging purposes the original message and the credentials are also stored. However, this is not part of the system's main functionality and is thus not depicted in Figure 1.

The *Validator* component uses a *Format Description* (currently DTD), to verify that the message conforms to the specified context data format. This provides a very flexible mechanism to find malformed messages early in the process and avoids the unnecessary use of resources on messages that may have been transmitted by outdated gathering mechanisms, or that may have been damaged. After the format has been verified, the *Context Listener* is informed about the validity of provided messages and in turn provides the mobile device with the success or corresponding error information and closes the connection to the mobile device.

Validated messages are passed to the *Parser* component, which in turn is responsible for extracting the context information from the message. The information is then provided to the *Context Repository* (see next section), together with a unique identifier

for the message that contained it. The different types of context information, alongside with the message identifier, are submitted to the Notification Queue component.

The *Notification Queue* implements a mechanism that allows different systems to subscribe to specific context information (e.g. based on type, source, etc.). This kind of event mechanism enables different reasoning mechanisms to subscribe to specific context information. As reasoning mechanisms are often processing intensive and may be time-consuming, the mechanism furthermore enables the invocation of several instances of the same mechanism, to process the data in a timely manner and to return results to the mobile device on time.

B. Data Management

The primary task of data management is handled by the *Context Repository* component. It implements a flexible mechanism that stores context data and provides it to other components. An important functionality of the repository is the adaptation of the storage mechanism to changes in the context data sets. As described earlier, the Validator component uses a Format Description to validate the correctness of incoming messages. The Context Repository uses the same format description as the Validator, to adapt the structure of the database. While this functionality is important and not necessarily trivial to implement, it is not depicted in Figure 1, as it does not represent the primary functionality of this component. The specific implementation is however described later.

In order to allow a user to control the provided data, a *Web Interface* component is responsible for providing an interface for the management of context data in the repository. This component enables each user to browse the data that (s)he has submitted, and to delete information where necessary. The component furthermore enables the remote administration of users and context data by authorized persons. The Web Interface component also uses the external Authentication mechanism, in order to authenticate users and administrators.

C. Context Reasoning

As mentioned earlier, the system should allow different tools and reasoning mechanisms to access

the context data. Although the reasoning itself is implemented outside the system, the different tools and mechanisms require means to adapt the provided information to their specific formats. For this reason, a *Data Export* component, or a set thereof, undertake the subscription to the Notification Queue component. When the data export components are informed about the availability of the subscribed context data, they can use the provided message identifier as means to obtain the actual context data from the Context Repository. The data export tools will then bring the data into such a format that it can be understood by external reasoning mechanisms. It is thereby possible that one such component implements the export functionality to several different formats. This component furthermore takes care of informing the reasoning mechanisms and, if necessary, triggering the actual reasoning process.

V. IMPLEMENTATION

While the previous section discussed the architecture of the system, this section provides further information regarding the actual implementation of the system.

A. Execution Environment

In order to minimize implementation effort, Java [14] is used, with J2EE [15] as the component model for the server.

In particular, due to the practical availability of the system, a JBoss Application server [16], running on the Linux servers of the Computer Science Department of the University of Helsinki was chosen, together with the following additional libraries:

- JAXB: generation of Java interfaces and implementation classes from a given DTD and JAXB configuration file [17].
- HyperJAXB: generation of Hibernate mappings for the generated JAXB interfaces from the given DTD and JAXB configuration file that contains Hyper-JAXB annotations [18].
- Hibernate DB Schema Export: generation of database schema from the Hibernate mappings, generated by HyperJAXB [19].
- Apache Axis: handling of SOAP messages [20].

B. Data Exchange

For the communication between the mobile client and the Context Server, SOAP calls on top of HTTP were chosen. While an HTTP stack is available on most high-end mobile services, its use reduces the implementation effort significantly, compared to other alternatives. The use of SOAP as the packaging protocol was also motivated by the existence of supporting libraries for both, the client, as well as the server side. Furthermore, *Document Type Definition* (DTD) was used to specify the format of valid messages, as described in the architecture section earlier. Many existing libraries are available that implement functionality necessary to handle DTD documents and provide the necessary information.

C. Data Storage

The data storage mechanism used for the implementation is based on the relational database mechanism that is part of JBoss. For logging purposes, each incoming message is stored in its original format, together with a time stamp and the identifier of the submitting terminal. Actual context data is retained only from authenticated users and validated messages.

As described earlier, the actual structure of the database is specified by the DTD. Depending on changes in this description, the structure of the database is adapted automatically, by creating new tables for structures that have changed or that have been added. The data of structures that are being replaced is stored in a backup, in order to be accessible later on.

D. Web Interface

As described earlier, the Web Interface component provides a mechanism for users, to manage submitted context data. Figure 2 shows a snapshot of the management interface.

A similar interface enables the administration of users and their context information by authorized persons. In order to ensure a basic level of system security, the user or administrator is required to authenticate her/himself, before either interface can be accessed.

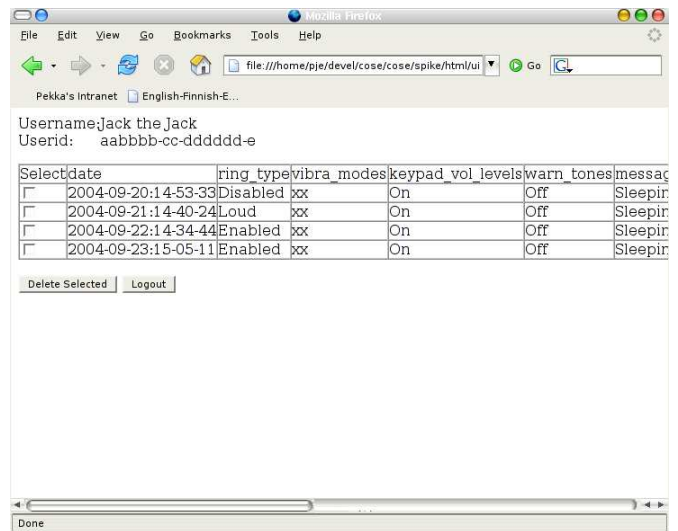


Fig. 2. Web Interface

E. Data Export

In its current implementation, the Data Export component can serialize the context data into two different formats. The Attribute-Relation File Format (ARFF) [21] is used by the WEKA reasoning toolkit [22], whereas the second format currently supported by the system is the flat file format (e.g. [23]).

F. Context Reasoning

In the current implementation the system is primarily used to verify the behavior of specific reasoning mechanisms, with the WEKA toolkit. After the Export tool has serialized the context data, it triggers the execution of a particular reasoning mechanism, using the APIs that are provided by WEKA. Of particular interest in our research are the following interfaces:

- Classifier
- Instance
- Attribute
- Filter (data pre-processing)

G. Testing

In order to test the functionality of the system, several test clients have been implemented. One test client is based on Perl, while a second client is implemented in Java. Both communicate with the system, using SOAP messages on top of HTTP. The clients transmit the provided messages and

inform the user about the results that are provided by the server. Alternatively, also a web interface is available, which allows the direct specification of context data and transmits it to the server.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we introduced the architecture and the implementation of a context gathering server, which enables remote context reasoning. The context data, gathered by this server, enables the testing and verification of specific context reasoning mechanisms on real-life data.

While the current implementation is functional, several extensions are being worked on. These include a more elaborate client on advanced mobile devices, which is able to gather a wider variety of context data.

Further research will also concentrate on the development and verification of new reasoning mechanisms, which may bring with it the need to implement additional components, or even to adapt the architecture of the system.

VII. ACKNOWLEDGMENTS

The authors would like to thank especially Otso Virtanen for his help in getting this project started and his valuable contributions during the implementation phase. We furthermore want to thank Antti Tevanlinna and the students of the University of Helsinki, who helped in the implementation of the prototype of this system as part of their software engineering project: Pekka Enberg, Laura Kataja, Oskari Ojala, Juha Ojaluoma, Miikka Sauramo and Timo Toikkanen.

REFERENCES

- [1] P. Nurmi and P. Floréen, "Online feature selection for contextual time series data," Presented in *PASCAL workshop on subspace, latent structure and feature selection techniques: statistical and optimisation perspectives workshop*, Feb. 2005, <http://www.cs.helsinki.fi/u/ptnurmi/papers/slsfs05.pdf>.
- [2] P. Nurmi, P. Floréen, M. Przybilski, and G. Linden, "A framework for distributed activity recognition in ubiquitous systems," in *Proceedings of the 2005 International Conference on Artificial Intelligence (ICAI '05)*, 2005, to appear.
- [3] M. Przybilski, P. Nurmi, and P. Floréen, "A framework for context reasoning systems," in *Proceedings of the 23rd IASTED International Conference on SOFTWARE ENGINEERING (SE 2005)*, 2005, pp. 448 – 452.
- [4] P. Nurmi, M. Przybilski, G. Lindén, and P. Floréen, "An architecture for distributed agent-based data preprocessing," in *Proceedings of the Workshop on Autonomous Intelligent Systems: Agents and Data Mining (AIS-ADM)*, ser. Lecture Notes in Computer Science, V. Gorodetsky, J.Liu, and V. Skormin, Eds., vol. 3505. Springer-Verlag, 2005, to appear.
- [5] M. Przybilski, "Distributed context reasoning for proactive systems," in *Proceedings of the International Workshop on Context Awareness for Proactive Systems (CAPS '05)*, 2005, to appear.
- [6] T. P. Moran and P. Dourish, "Introduction to special issue on context-aware computing," *Human-Computer Interaction (HCI)*, vol. 16, no. 2-3, pp. 87 – 96, 2001.
- [7] M. Wallbaum and P. Dornbusch, "Design considerations for a platform supporting location-aware services," in *Proceedings of Multimedia, Internet Video Technologies (MIV)*, 2001.
- [8] T. Winograd, "Architectures for context," *Human-Computer Interaction (HCI)*, vol. 16, no. 2-4, pp. 401 – 419, 2001.
- [9] S. S. Yau, F. Karim, Y. Wang, B. Wang, and S. K. Gupta, "Re-configurable context-sensitive middleware for pervasive computing," *Pervasive Computing*, vol. 1, no. 3, pp. 33–40, Jul.-Sep. 2002.
- [10] H. A. Duran-Limon, G. S. Blair, A. Friday, P. Grace, G. Samartzidis, T. Sivaharan, and M. Wu, "Context-aware middleware for pervasive and ad hoc environments," 2003.
- [11] P. Korpipää, J. Mäntyjärvi, J. Kela, H. Keränen, and E.-J. Malm, "Managing context information in mobile devices," *Pervasive Computing*, vol. 2, no. 3, pp. 42 – 51, Jul. - Sep. 2003.
- [12] A. K. Dey, G. D. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interaction (HCI)*, vol. 16, no. 2,3&4, pp. 97 – 166, 2001.
- [13] T. Hofer, W. Schwinger, M. Pichler, and G. Leonhartsberger, "Context-awareness on mobile devices, the Hydrogen approach," in *Proceedings of the 36th International Hawaii International Conference on System Sciences (HICSS'02)*, 2002.
- [14] Sun Microsystems, Inc., "Java Technology," <http://java.sun.com>, 2005.
- [15] —, "Java 2 Platform, Enterprise Edition (J2EE)," <http://java.sun.com/j2ee>, 2005.
- [16] JBoss Inc., "JBoss Application Server," <http://www.jboss.org/products/jbossas>, 2005.
- [17] Sun Microsystems, Inc., "Java Architecture for XML Binding (JAXB)," <http://java.sun.com/xml/jaxb/>, 2005.
- [18] HyperJAXB Project, "HyperJAXB Project," <https://hyperjaxb.dev.java.net/>, 2005.
- [19] Hibernate, "Introduction to Hibernate," <http://www.retroevolution.com/hibernate.htm>, 2005.
- [20] The Axis Development Team, "WebServices - Axis," <http://ws.apache.org/axis/>, 2005.
- [21] G. Paynter, "Attribute-relation file format (ARFF)," <http://www.cs.waikato.ac.nz/ml/weka/arff.html>, 2004.
- [22] University of Waikato, "WEKA Homepage," <http://www.cs.waikato.ac.nz/ml/weka/index.html>.
- [23] Helsinki Institute for Information Technology (HIIT), Complex Systems Computation Group (CoSCo), "Data format specification [B-Course]," <http://b-course.hiit.fi/format.html>, 2002.