Using Web Services to Build Context-Aware Applications in Ubiquitous Computing

Gerhard Austaller, Jussi Kangasharju, Max Mühlhäuser

Telecooperation Group
Department of Computer Science
Darmstadt University of Technology
64289 Darmstadt, Germany,
`{gerhard,jussi,max}@tk.informatik.tu-darmstadt.de`

Abstract. Ubiquitous and mobile web applications are typically very autonomous in nature, because they rely on additional information about the user's context. In this paper we present a general context model for including context information into ubiquitous and mobile web applications. Our model is based on layers, which cover the path from context sources to the application level, including all intermediate filtering and context fusion. As an example, we present a context-aware calendar application built according to our context model.

1 Introduction

The credo of ubiquitous computing is that hardware and software in everyday life should "disappear", and be as autonomous as possible. This requirement has large effects on the system design. Autonomy can be achieved through the use of additional information about the user's context. Our definition of context information follows that of Dey [1] and Schmidt [2], in that any relevant information about the user's situation can be considered to be context. In addition to this, we also consider as context information sources which are not in the user's immediate vicinity, but contain information that is relevant to the user at that time.

In this paper, we develop a general model for including context information into mobile and ubiquitous applications. Our model has several layers, from the context sources to the application, which perform all the tasks of context filtering and fusion, so that the application only receives the relevant information about the user's context.

As an example application of our model, we implement a context-aware calendar which keeps track of a user's appointments and reminds her when she should leave for her next appointment; this reminder naturally depends on where the user is, where the next appointment is, etc. In order to be able to exploit as many existing sources of context, we have decided to built our calendar using web services.

Hence, the contribution of this paper is two-fold. First, we present our general context model for building context-aware applications and services. Second, we present an implementation of our context model using a web services architecture.

This paper is organized as follows. Section 2 presents the context layers we propose and section 3 discusses the implementation of the context-aware calendar. We review related work in Section 4. Finally, Section 5 concludes the paper.

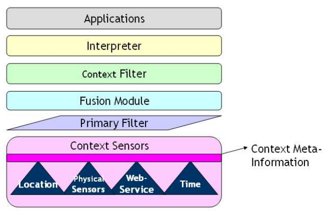

Fig. 1. Context Stack

2 Context Model

This section describes to conceptual model of our context stack. By introducing several layers of abstraction between the actual context source and the application, we allow programmers to use context information in their applications in an easy and straightforward manner. Such layered structures are widely used in programming (e.g., graphics libraries, TCP/IP network stack), but have only recently been considered by researchers in context aware computing [1]. Figure 1 shows the layers in our general context model. The rest of this section discusses the details of each layer.

Context Sensors

The lowest layer is the *Context Sensors* layer. This layer contains all the context sources which are of interest to us. Regardless of the name, we do not limit ourselves to actual physical sensors as context sources, such as temperature, light intensity, or noise level sensors. Context information such as calendar appointments or timetables are typically considered higher-level context information; however, we include them in this layer as well.

This layer feeds context information to the rest of the stack. It is responsible for expressing the context information in a suitable format and also adds metadata (e.g., a timestamp) to the information. This metadata is passed up along the stack so that the higher layers can decide whether that piece of information is still relevant.

Primary Filter

The next layer is the *Primary Filter* layer. Its main purpose is to protect the stack from being flooded by information. When context information is continuosly delivered from a source (e.g., a temperature reading), the application may be interested in only a few values every now and then. This is especially important for context sources which deliver their information by pushing it, instead of the application pulling it.

Fusion Module

The role of the *Fusion Module* is to fusion the context delivered through the Primary Filter. This fusioning can either make the context information easier to access (for the higher layers) or to get a higher-level representation of the raw context information. Some abstracting functionality is also included in the next higher layer, the Context Filter (see below).

An example of how the Fusion Module can provide easier access to context information is a positioning system. Consider a badge-based positioning system which is able to locate users at the granularity of a room. The Fusion Module can collect information from all sensors in all rooms and present a view to the higher layers which allows querying users. In this case, the Fusion Module only facilitates access to context information and does not change any information.

Context Filter

The role of the Context Filter is similar to the Primary Filter in that it filters information coming from the Fusion Module. However, the Context Filter acts on a higher level and is therefore able to perform "intelligent" filtering. For example, a positioning system may send periodic events indicating which user is in which room. The Context Filter takes these events and is able to generate higher-order events from them, such as "person X entered room Y".

Interpreter

The Context Filter only handles context of one kind while the Interpreter can deal with different sources of different kinds of context. This is important for higher-level reasoning and implying events which cannot be measured. An example is a meeting taking place in a room. In this case, there are many people in the room, the noise level indicates people speaking, and the lights are turned on. None of the individual context "readings" alone is sufficient to infer a meeting taking place, but put together, the Interpreter can deliver us this result.

Application

At the top of the stack are the applications using context. There are many possibilities to use context in applications. Context awareness is an enabling technology for new kinds of applications. It helps increase autonomy of applications, including adaption of behaviour and user interface.

3 Implementation of a Context-Aware Calendar

Our goal in implementing the context-aware calendar was to validate our context model, test the usability of such a calendar in everyday life, and to evaluate the suitability of web services for ubiquitous applications. Web services provide an open architecture where all service providers can freely compete. We believe this gives them a significant advantage over intergrated solutions where one provider is responsible for providing all services, with no major incentive to innovate. Good examples of the successes of open architectures are the Internet and the Web, as well as the iMode-service from NTT DoCoMo (as compared to the European WAP services, which represent more the integrated model).

Overview of the Calendar Application

The Coordinator is the core of the context-aware calendar. It runs the algorithms for service discovery, planning the notifications, and sending notifications. Planning is done by gathering information from several sources, the appointments database, a public transport timetable, and a positioning system. When the user should leave for her next appointment, the coordinator sends a notification to the user as a text message.

Appointments

The source for the actual calendar appointments was the Microsoft Exchange server for our group. The rather limited reminder possibilities offered by the Exchange server are replaced by our context-aware calendar. (The limited reminders in Exchange were

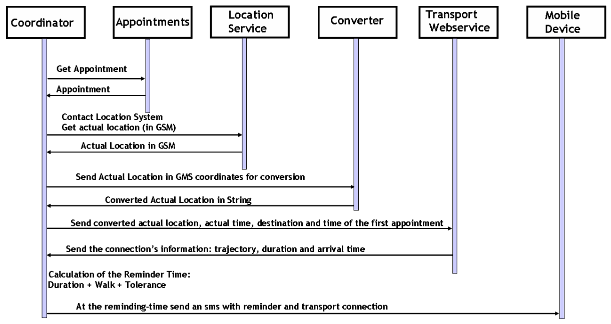
Coordinator
Appointments
Location Service
Converter
Transport Webservice
Mobile Device
Get Appointment
Appointment
Contact Location System
Get actual location (in GSM)
Actual Location in GSM
Send Actual Location in GMS coordinates for conversion
Converted Actual Location in String
Send converted actual location, actual time, destination and time of the first appointment
Send the connection's information: trajectory, duration and arrival time
Calculation of the Reminder Time:
Duration + Walk + Tolerance
At the reminding-time send an sms with reminder and transport connection

Fig. 2. Coordinator and its functionality.

actually a strong motivating factor for us choosing to implement a calendar as an example of our context model.)

Public Transport Timetable

We wrote a Java wrapper for accessing the information from the public transport timetables. The wrapper realizes a web service for querying any kind of timetables by separating the web service specific code from the HTML parser. Our wrapper presents an interface, specified in a WSDL document, which allows queries for getting from place A to place B. The interface returns the means of transport to take, how and where to change, and the total time of traveling.

Positioning System

For determining the current location of the user, we use the "O2 Handy Finder" provided by the operator O_2The "Handy Finder" website returns a map as an image on which the user's location is plotted. The website also returns the coordinates as text in an HTTP-header. The coordinates express the longitude and latitude of the base station under which the user's mobile phone is currently located.

Coordinator

The core of the context aware calendar is the coordinator. The coordinator discovers the feasible services, coordinates the message exchange, and implements the planning algorithm. The coordinator is illustrated in Figure 2. The planning algorithm is at the heart of the coordinator. This algorithm is responsible for keeping the context information up-to-date and deciding when to send out the notification to the user.

Notification to the User

We use the standard text messaging (SMS) of GSM mobile phones for sending notifications to the user. The main advantages of SMS are two-fold. First, mobile phones are ubiquitous, hence most users already possess a device for receiving the notifications. Second, SMS is a push-based technology which is required of a notification service. Feedback from the phone back to the calendar is part of our future work.

4 Related Work

There has been lot of work in the area of context-aware computing. First applications include the phone forwarding application built on top of the Active Badge Location

System [4] and the GUIDE project [3]. With the upcoming mobility of devices, work such as [7] explored the efficient distribution of notifications to clients. Dey [1] introduced the separation of concerns and reusage in the design process of context aware systems. Later models include [2] and [8]. Recent work has studied automatic building of context models [9] or predicting context [10].

In contrast to the previous work, our context model explicitly formalizes the different layers between the context sources and the application. Formalizing the context model in layers gives us the advantage that we can use existing modules for each layer and plug in the appropriate modules or layers on-demand. Furthermore, our layered model, based on ubiquitous services, is very scalable, and can handle a large number of context sources, filter, and context consumers (applications) concurrently.

5 Conclusion

In this paper we presented a general context model for including context information into ubiquitous and mobile web applications. Our model contains 5 layers: context sensors, primary filter, fusion module, context filter, and interpreter. As an example, we presented a context-aware calendar application built according to our context model. We have built our calendar using web services, in order to take advantage of the many context sources on the Web, and because it makes our application open and easily extensible. We also discussed the usefulness of web services in ubiquitous and mobile web applications.

References

1. Dey, A.K.: Providing Architectural Support for Building Context-Aware Applications. PhD thesis, Georgia Institute of Technology (2000)
2. Schmidt, A., Gellersen, H.W.: Modell, Architektur und Plattform für Informationssysteme mit Kontextbezug. Informatik Forschung und Entwicklung **16** (2001)
3. Davies, N., Mitchell, K., Cheverst, K., Blair, G.: Developing a context sensitive tourist guide. In: First Workshop on Human Computer Interaction for Mobile Devices. (1998)
4. Want, R., Hopper, A., Falcão, V., Gibbons, J.: The Active Badge Location System. ACM Transactions on Information Systems **10** (1992) 91–102
5. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Achieving scalability and expressiveness in an internet-scale event notification service. In: ACM Symposium on Principles of Distributed Computing, Portland, OR (2000)
6. Kügler, D., Vogt, H.: Marking: A privacy protecting approach against blackmailing. In: Public Key Cryptography PKC 2001, Cheju Island, Korea (2001)
7. Schilit, W.N.: A System Architecture for Context-Aware Mobile Computing. PhD thesis, Columbia University, New York, NY, US (1995)
8. Großmann, M., Leonhardi, A., Mitschang, B., Rothermel, K.: A world model for location-aware systems. Informatik **8** (2001) 22–25
9. Takada, T., Kurihara, S., Hirotsu, T., Sugawara, T.: Proximity mining: Finding proximity using sensor data history. In: IEEE Workshop on Mobile Computing Systems and Applications, Monterey, CA (2003)
10. Petzold, J., Bagci, F., Trumler, W., Ungerer, T.: The state predictor method for context prediction. In: UbiComp 2003: Adjunct Proceedings. (2003)