The Design and Use of a Generic Context Server

Daniel Salber and Gregory D. Abowd
GVU Center, College of Computing
Georgia Institute of Technology
801 Atlantic Drive, Atlanta, GA 30332-0280
{salber, abowd}@cc.gatech.edu

Abstract

Although context-awareness is a key component for perceptual user interfaces, we lack generic infrastructures for developing context-aware applications. We propose a generic infrastructure based on context servers that store, share and archive contextual data. We describe a few applications we have built that take advantage of context sharing and context history and then turn to the overall design of our context server.

1.Introduction

Context-awareness is recognized as an important feature for ubiquitous and wearable computing. Context sensing and interpretation techniques are maturing and applications demonstrate the value of using context. However, bridging the gap between sensing and interpretation techniques on one hand and applications on the other hand is mostly done using ad hoc techniques. The lack of generic infrastructure requires developers to rely on custom solutions for handling context and hinders the development of new applications.

In this paper, we first look at common sources of context and emphasize the need for two often overlooked generic context handling features: context sharing in multi-user settings and context history. We describe applications that take advantage of these features. We then turn to the design of an infrastructure for supporting such applications. We first explain our goals and then describe our architecture for a generic context server.

1.1.What is Context?

Context covers information that is part of an application's operating environment and that can be sensed by the application. This typically includes the location, identity, activity and state of people, groups and objects. Places such as buildings and rooms can be fitted with sensors that provide measurements of physical variables such as temperature or lighting. Finally, an application may sense its software and hardware environment to detect, for example, the capabilities of nearby resources. Context-aware applications sense context information and modify their behavior accordingly without explicit user intervention.

In the next section, we look at commonly used sources of context. Besides these, we identify two overlooked areas: sharing context in multi-user settings and exploiting context history.

1.2.Sources of Context

We distinguish four broad categories of context: physical, system, application, and social. The physical context and notably the user's location is a most commonly used source of context [1, 4, 9, 11]. The system's environment is a second popular source of context: nearby computing resources [12] or network traffic [16] are presented to the user or used to tailor the interaction. Similarly, application data, such as the current text selection, provides context information that can help anticipate user actions [3, 5, 10]. Finally, the social environment is a relevant source of context information: information about people such as the presence of people, their identity, their activity, may be used by systems to either provide information to other users or tailor the system's behavior to a user's needs or preferences [1].

Although research has been carried out on identifying and tracking people in a scene, there are few convincing applications that take advantage of social context. Sharing context information can provide richer social context and provide a foundation for applications. We also observe that context-aware applications use context information at the present time. Context history also provides interesting information. Still, sharing context and context history are two little explored areas.

1.3.Sharing Context

There is previous work on sharing physical context information and namely location. For instance, a user's current location tracked by an active badge may not trigger any relevant actions for the wearer. But this information will help a colleague locate her to discuss an urgent problem or allow the secretary to forward a phone call [15]. However, context sharing can be extended profitably to a user's system and application context in order to provide social context to other users. The applications described in section 2 rely on that form of context sharing. Context

sharing at the level of a group is another unexplored area.

An application we envision provides a group of people gathered in a social area with a display of news likely to interest most of them. By gathering each user's preferred daily Web sources of news, it decides to display on a large screen the news source that most people present prefer and the news items people haven't read yet. Another potential application relies on the inspection of people's unread email. When a member of a workgroup sends everybody else an urgent message (e.g., a meeting time change), the application allows the sender to check that everybody has read the message.

In group-level context sharing, context information from several users is gathered and synthesized into a new piece of context. Everybody being aware of the meeting time change is social context to the sender. This gathering of private information may appear like a potential privacy threat. To give users control over the information that is gathered, our infrastructure provides users with customizable privacy protection mechanisms.

1.4.Context History

Most context-aware applications deal with context data concerning the present. Except for context-based retrieval applications [6], there has not been much work done on the value of context data history. A marginal example is capture applications. In this case, context data is not remembered for itself, but because it is associated with some captured piece of data.

However, in everyday social relationships we rely naturally on historical data. For instance, when looking for somebody, we ask colleagues if they have seen this person in the recent past. This information may actually be more useful than the current location of the person. If she's been in her office, she saw the note left on her desk, or she's probably read her email. A user's recent context (her whereabouts) helps other users interact with her.

Similarly, a history of URLs visited may provide interesting clues to both the user and colleagues as to what tasks the user was engaged in.

Another example would be a context history-aware note-taking aid. It could look up meeting history information and pull up notes taken the last time the user was attending a meeting with the same persons. A similar application could be provided to students attending classes.

2.Applications

We have designed and built applications based on a generic context server. In this section, we describe three of them that exploit context sharing and context history capabilities of the infrastructure.

2.1.Where Have You Browsed Today?

The "Where Have You Browsed Today?" application aims at stimulating discussion between people who may share common interests based on their web surfing activity. In contrast to collaborative browsing tools [8], this application matches users' interests after they're done browsing to stimulate interaction when they may be more available to engage in discussion. It uses both context history and shared context information.

The application consists of a URLs logger that captures the current URL displayed in the user's web browser. Using the history feature, logs of visited URLs can be generated. At the end of the day, URLs logs are compared for common web pages or sites. The result is used to notify the users if they've been visiting the same pages or sites that day.

It is important to note that users do not know each other's URL logging history, which most users would consider private data. Only those URLs that are common to all users are revealed and only to them. Still, other options may be explored, like requesting permission from each user before sharing her common URLs.

2.2.Are You Reading Me?

The "Are You Reading Me?" application uses context sharing to facilitate email communication. Suppose Daniel needs to send an urgent message to Gregory, who is usually overloaded with email. Email seems convenient but is it the right medium to get in touch effectively with Gregory today?

The "Are You Reading Me" application adds two functions to Daniel's email client:

- The first one allows Daniel to know how many messages are left unread in Gregory's Inbox.
- The second function allows Daniel to know how many previous messages from him to Gregory are still left unread.

Daniel's email client fetches these two pieces of information from Gregory's context. They allow Daniel to assess Gregory's current email load and the fitness of email for sending an urgent message.

Gregory has the possibility to restrict access to these pieces of his context. Typically one would want only close colleagues to be able to inquire about one's current email load.

2.3.Let's Have A Meeting!

The "Let's Have A Meeting!" application uses context sharing to provide more efficient scheduling. When two people decide to have a meeting, they usually both create an entry into their schedule. Both entries have the same date, symmetrical information (A enters "meeting with B", B enters "meeting with A") and each user may add private notes. Using context may alleviate this duplication of work.

With our application, only one user has to create an entry in her schedule. She then shares this entry with the other person involved. If for instance, Gregory and Daniel decide to schedule a meeting, Gregory creates an entry labeled "meeting with Daniel" in his schedule. With an extra click, he shares this entry with Daniel. This action creates a symmetrical entry (i.e., "meeting with Gregory") in Daniel's schedule at the same date. If the meeting involves a third party, the name of the third party appears in both entries. Users can add personal information to the entry once it is created.

In this case, both Daniel's and Gregory's context information is used. When Gregory shares his meeting entry, his own context is queried for the user's name to reconstruct a complete meeting entry, namely to add Gregory's name as a participant to the meeting (this was implicit in Gregory's entry). Then, the complete meeting information is sent over to Daniel's scheduling application. This application in turn queries Daniel's context for the user's name and scans the meeting information to try to match the user's name. It then removes it if present and creates the entry in Daniel's schedule.

3. Context Infrastructure Design

In this section, we describe our design of a context infrastructure to support the applications described in the previous section. We first outline our design goals and observe that existing context infrastructures don't achieve them. We then turn to our overall design and architecture.

3.1. Design Goals

To provide the services required by the applications we just described, our infrastructure goals are threefold:

1) allow for networked applications to access local and remote context data in a heterogeneous environment;
2) accommodate a variety of applications, sensors, and operations on context data;
3) preserve the history of contextual data sensed.

With these objectives in mind, let us examine previous work on context infrastructures.

3.2. Previous Work

A few context-handling infrastructures have already been developed notably Schilit's architecture for context-aware mobile computing [12] and Hull *et al.*'s SitComp service [7].

Schilit's architecture mainly aims at storing context data in a repository accessible by networked applications running on mobile ParcTab devices or fixed computers. Applications as well as sensors manipulate context data directly and thus must be aware of the storage model. Independence of applications and sensors from context data is not guaranteed. Furthermore, no provision is made for storing the history of context data.

The SitComp (Situated Computing) service software component utilizes local sensors to provide situation information to applications through an API. Applications can either query the SitComp service or ask to be notified of context changes. SitComp also performs fusion of data from multiple local sensors and abstracts raw data into context information at a higher level of abstraction. SitComp puts the emphasis on these abstraction mechanisms and provides a clear separation between applications on one hand and context sensing and abstracting mechanisms on the other. However, SitComp is intended for applications running on a single computer and doesn't allow remote access. Although the authors envision using context history for context-based retrieval, SitComp does not seem to support this yet.

3.3. Global Design

Our infrastructure is comprised of context servers that maintain a dynamic model of context data. We look at the services provided by context servers and assess how this infrastructure achieves our objectives stated in section 3.1.

In our model, context data is sensed by devices and gathered in a repository by the computer these devices are attached to. A computer is attached either to persons (individuals or groups) or places (e.g., rooms, buildings, vehicles). Each computer runs a context server that performs context management and provides access to context data: it acquires raw local context data through sensors and stores it and provides access to this data to local and remote applications. In addition, context servers run services, called context synthesizers, that act on local and/or remote context data to generate context information at a higher level of abstraction (see Figure 1).

Access to local and remote context data is provided through a context access API. This API guarantees independence of applications from sensors as well as from the particular storage model used. To allow access by heterogeneous clients, the API is a network API based on XML [14] and HTTP. Each context server runs an HTTP server as well as an XML parser. Requests and replies are encoded in XML. As in SitComp, two models of communication are supported: events and requests. In the events-based mechanism, the context-generating component (either a sensor or the context repository) generates events to registered components when the sensed or stored data changes. In the request method, the context gathering component (either an application or the repository) polls the context-generating component when needed. This distinction reflects the dichotomy already observed in user interfaces between status, i.e.,

continuously available information and events, i.e., atomic, transient information [2].

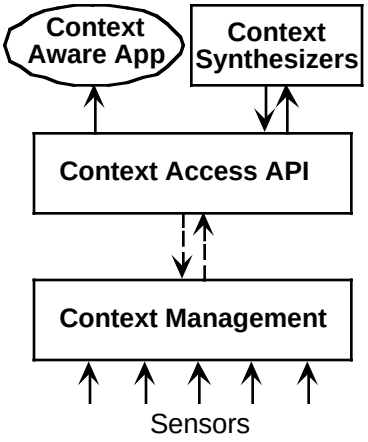

Figure 1. The architecture of the context server. The three rectangles constitute the context server. The context management component handles context acquisition and storage. The context API provides local and remote access to context data. Context synthesizers abstract raw context data into higher level information. Arrows show context data flow between components. Dashed arrows denote XML encoded communications.

Raw context data must often be abstracted into higher level information. To achieve this, a context server hosts synthesizers. Synthesizers are pluggable modules that access context data through the API and generate new context data that is fed back to the context management component. Examples of abstraction mechanisms include: deducing the state and country from a city name and assessing if a room is occupied or not by combining ambient lighting, sound level and presence sensors data. Synthesizers also aggregate context data from multiple context servers and perform comparisons as in the URLs log comparison example described in 2.1. Another example of aggregation is the detection of spatial relationships like adjacency or inclusion between geographical context information collected from multiple context servers.

Finally, to allow the use of context history, context servers log context data changes and preserve historical data. They allow access to context data at any point in the past or retrieval of data over a time interval.

The architecture we presented is generic: Context servers are not tied to applications or sensors. The context access API ensures independence from applications. Sensor-specific components are insulated from the context management layer and can be reconfigured dynamically to suit the context needs of applications.

The current context server prototype is implemented in Frontier, a scripting language and environment that runs on MacOS and Windows [13]. Frontier provides us with a persistent object database, XML encoding and parsing, HTTP client and server support, and an AppleEvents or COM interface for inter-application communication.

4. Conclusion

We have presented a generic context-handling infrastructure based on context servers. Context servers are particularly suited for sharing context data and providing access to context history. These features enable us to explore promising new applications. Our immediate goal is to develop more applications based on the context servers infrastructure. Of particular interest are applications that rely on elaborate context interpretation and applications aimed at mobile users. More information is available from: http://www.cc.gatech.edu/fce/contextserver/

5. Acknowledgments

The first author is currently funded by a fellowship from INRIA. We wish to thank members of the Future Computing Environments group at Georgia Tech for fruitful discussions and particularly Anind Dey for insights and comments.

6. References

[1] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper and M. Pinkerton. Cyberguide: A Mobile Context-Aware Tour Guide. *ACM Wireless Networks*, 3:421-433, 1997.

[2] G. D. Abowd and A. J. Dix. Integrating status and event phenomena in formal specifications of interactive systems. *ACM Software Engineering Notes*, 19(5):44-52, December 1994.

[3] Apple Research Laboratories. *Apple Data Detectors homepage*. http://www.research.apple.com/research/tech/AppleDataDetectors/, Apple Computer, 1997.

[4] N. Davies, K. Mitchell, K. Cheverst and G. Blair. Developing a Context Sensitive Tour Guide. *Proceedings of First Workshop on Human-Computer Interaction for Mobile Devices*, pp. 64-68, 1998, Glasgow, UK.

[5] A. Dey. Context-Aware Computing: The CyberDesk Project. *Proceedings of the 1998 Spring AAAI Symposium on Intelligent Environments*, 1998.

[6] M. L. M. Flynn. Forget-me-not: Intimate computing in support of human memory. *Proceedings of FRIEND21: International Symposium on Next Generation Human Interfaces*, pp. 125-128, 1994.

[7] R. Hull, P. Neaves and J. Bedrod-Roberts. Towards Situated Computing. *Proceedings of IEEE ISWC'97, First International Symposium on Wearable Computers*1997, Cambridge, MA, USA.

[8] H. Lieberman, N. V. Dyke and A. Vivacqua. *Let's Browse: A Collaborative Web Browsing Agent*. http://lieber.www.media.mit.edu/people/lieber/Lieberary/Lets-Browse/Lets-Browse.html, MIT Media Lab, 1998.

[9] E. D. Mynatt, M. Back, R. Want and R. Frederick. Audio Aura: Light-Weight Audio Augmented Reality. *Proceedings of the ACM UIST'97 Symposium on User Interface Software and Technology*, p. 211-212, 1997.

[10] M. Pandit and S. Kalbag. The Selection Recognition Agent: Instant Access to Relevant Information and Operations. *Proceedings of Intelligent User Interfaces '97*, 1997.

[11] J. Pascoe, N. Ryan and D. Morse. Human-Computer-Giraffe Interaction: HCI in the Field. *Proceedings of First Workshop on Human-Computer Interaction for Mobile Devices*, pp. 48-57, 1998, Glasgow, UK.

[12] W. N. Schilit. *System architecture for context-aware mobile computing.* Ph.D. Thesis, 1995, Columbia University.

[13] UserLand Software. *Frontier 5.1*. http://www.scripting.com/frontier5/, UserLand Software, 1998.

[14] W3C XML Working Group. *Extensible Markup Language (XML) 1.0.* http://www.w3.org/TR/1998/REC-xml-19980210, World-Wide Web Consortium, 1998.

[15] R. Want, A. Hopper, V. Falcao and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 10(1):91-102, 1992.

[16] M. Weiser and J. S. Brown. Designing Calm Technology. *Workshop on Ubiquitous Computing at CHI 1997*, 1997.