CONTEXT-AWARE COMPUTING USING A SHARED CONTEXTUAL INFORMATION SERVICE

Nancy Miller, Glenn Judd, Urs Hengartner, Fabien Gandon, Peter Steenkiste*, I-Heng Meng, Ming-Whei Feng, Norman Sadeh

Abstract
The Aura ubiquitous computing project is investigating how we can reduce user distractions by having applications automatically adapt to the user's context. Context-aware applications rely on a shared service, the Contextual Information Service, to obtain context information. In this paper we describe our experience in implementing four very different applications using the CIS and in porting the applications to a different environment. One of the services also integrates technologies developed by a sister project that focuses on using the Semantic Web to support context awareness and privacy.

1. Introduction

Fueled by advances in processing power, storage capabilities, and I/O devices, the proliferation of mobile computing devices is rapidly turning the focus of the computing away from personal computer and towards a collaboration between mobile devices, personal computers, and services. Unfortunately, the increased usage of mobile devices has also increased the amount of user effort required to operate these devices. This extra source of distractions can easily outweigh any benefit the user can gain from ubquitous computing. The Aura project [2] at Carnegie Mellon University is investigating how we can reduce user distractions by having applications automatically adapt to the user's context. Adaptation can be reactive or proactive, i.e. anticipating the user's needs.

An important design question in pervasive computing is how applications obtain the context information that they need to implement adaptive behavior. A simple ad hoc solution is to have each application independently collect the information it needs. Unfortunately this places a heavy burden on the application developer, making application development and maintenance time-consuming and expensive. Moreover, running the application in different environments requires a porting effort. In contrast, the Aura project relies on a shared service, called the Contextual Information Service or CIS [8], that collects context information for use by context-aware applications. We hoped to achieve two goals with this design. First, by levering the CIS, we hoped that application development would become significantly easier. Second, we hoped to improve application portability in the sense that once the CIS was available in a new environment, context-aware applications should run automatically.

*School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, prs@cs.cmu.edu

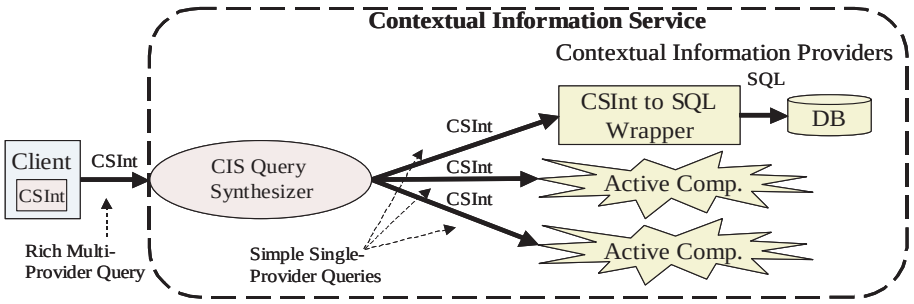

Figure 1. The CIS architecture

In this paper we report on our experience implementing four different context-aware applications that use the CIS for obtaining context information. This includes integrating our CIS module with a Semantic Web front-end developed by the MyCampus project that uses the Semantic Web in support of context awareness and privacy [1]. We also report on our experience in porting the CIS and applications to the Institute for Information Industry (III) in Taiwan as part of a collaborative project.

2. The Aura Contextual Information Service

Designing a contextual information service is a difficult task, both because of the diversity of the information and the complexity of the queries that must be supported. Queries can range from: "Where is the nearest color printer?", to "How much wireless bandwidth is likely to be available in room 213 at noon tomorrow?". The simplest approach to providing this information is to write custom contextual information services, as needed. Unfortunately, such an ad hoc approach will result in many, possibly overlapping, services with different interfaces, which will complicate application development. An alternative is to use a database. Databases provide well-understood mechanisms to structure data and make it accessible through a single consistent interface. Unfortunately, a static database precludes on-demand data collection, which is essential for dynamic information, such as a person's location. Moreover, contextual information often has meta-data associated with it (such as accuracy and precision) and databases do not provide support for this.

Our approach is to implement the CIS as a *virtual database*. Its interface is based on the widely used SQL database query language. However, the CIS is implemented as a distributed infrastructure that consists of contextual information providers that either store information or collect information on-demand. For example, people location information may be collected on demands, while information on printers and other devices can be stored in an actual database. A CIS query synthesizer is responsible for partitioning user queries into "subqueries" that can be handled by the information providers (see Figure 1). The different components communicate using a common Contextual Services Interface (CSInt), which is implemented as an XML-over-HTTP protocol. The CIS provides information about devices, people, networks, and physical spaces, and also about the relationships between these entities (Figure 2). More information on the CIS can be found elsewhere [8, 7, 6, 5, 4].

3. Adaptive and Proactive Applications

We now describe four context-aware applications that were built on top of the CIS.

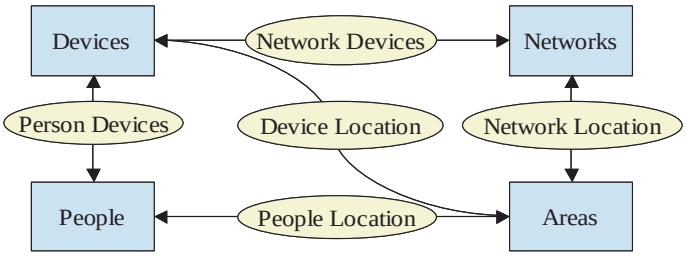

Figure 2. CIS provider classes

3.1. Nearest Printer

The Nearest Printer Service simply prints a document on the printer that is physically closest to the user. Besides proximity, additional metrics can be used to guide the printer selection. In our case, we restrict the search to public printers that have a relatively short print queue. Other metrics, e.g. resolution, print speed, or color, could be added. The Nearest Printer Service is an example of a broader class of "Nearest X" services, where X can be a type of device (e.g. projector) or building feature (e.g. restroom, coffee machine).

Given the availability of the CIS, this service is relatively easy to implement. It first uses the People Location Service to find out where the user is. Next, it uses the Device Service to identify suitable printers and the Space Service to evaluate proximity. The Device Service is implementated as a simple database since device features are mostly static. The printer queue length is however a dynamic property. For practical reasons (large number of printers and the high rate at which the printer queue can change), that information is obtained on demand using a separate service.

3.2. Nearest Conference Room

While this is an example of the "Nearest X" Service, we describe it explicitly because it has some unique features. When people are looking for a conference room, what they really need is an *available* conference room. Our service relies on two sources of information to help decide on availability: its reservation status and whether the room is currently occupied. We obtain the reservation status from the departmental conference room reservation system and we rely on X10 motion detectors for occupancy information. The service returns a short list of nearby conference rooms and their reservation and occupancy status; the user can decide (in part based on social conventions) which room they want to use.

The implementation of the Nearest Conference Room Service is similar to that of the Nearest Printer Service: it finds out where the user is and then uses the Space Service to identify appropriate conference rooms. The core of the Space Services is a database that contains building information organized in a hierarchical fashion [6]. To deal with the dynamic room status information, we rely on a separate Room Service that collects reservation and occupancy status on demand. This approach guarantees that the user is given fresh information and it also reduces the load on the Space Service database.

3.3. Context-Aware Messaging

Typically, it is the sender who determines how a message is delivered to the recipient (e.g. e-mail, voice mail, ..). However, in practice, it is the recipient who is most affected by the choice of delivery channel. An inappropriate channel can, on one hand, result in late delivery of an urgent message, or, on the other hand, an interruption by an inappropriate message result in an unnecessary distraction.

The Context-aware Message Delivery Service accepts messages from senders in a number of formats (input channels) and choses the message delivery mechanism (output channel) based on the user's preferences and current context. For example, a user can specify that, while he is attending presentations or lectures, all messages should be delivered using e-mail, except for urgent messages, which should be delivered using SMS. In the our initial prototype, the Context-aware Messaging Service accepts messages through e-mail and a custom web interface and it can deliver messages using: e-mail, SMS, and instant messaging. Other channels can easily be added, e.g. fax, voice mail, ..

Due to the wide variety of preferences users may have when it comes to specifying how they want different types of messages delivered to them under different contexts, this service was developed as a Semantic Web module that acts as a front-end to the CIS. The module is based on a customized version of the MyCampus Semantic Web engine [1]. It enables users to specify a rich set of message delivery preferences that relate to any combination of relevant contextual attributes. These attributes themselves are defined in relation to domain ontologies, namely domain descriptions that include the definition of relevant concepts and the relations among these concepts (e.g. calendar activity ontology, location ontology, delivery channel ontology, etc.). Message delivery preferences can be edited by users. They are saved in an extension of the OWL Semantic Web language [10] and loaded as decision rules into the Semantic Web engine along with relevant facts and ontologies.

The arrival of a new message is modeled as a new fact and results in the activation of one or more rules. In particular, the Semantic Web module relies on a set of service invocation rules that map contextual attributes it needs to process the incoming message onto queries of the CIS module and possibly other relevant sources of contextual information. Suppose the user specifies that, as long as he is not giving a presentation, all high-priority messages should be instant-messaged ("IM-ed") on his laptop. Upon receiving a high-priority message, the Semantic Web module activates the CIS service and requests the user's current activity. If it is determined not to be an instance of a presentation, the message is IM-ed to the user's laptop. Otherwise, the Semantic Web module goes on and looks for other relevant decision rules, in the process possibly sending other requests for contextual information to the CIS.

3.4. Proactive Task Assistant

The applications described so far consider the user's *current* context to optimize their operation. A proactive application, on the other hand, tries to anticipate the user's future needs, e.g. it acts based on the user's expected future context. Proactivity is much harder than (reactive) adaptivity in part because predicting context is hard. As a first step towards exploring proactivity, we developed a simple Proactive Task Assistant that helps users by automatically performing routine tasks automatically. Our Proactive Task Assistant leverages the idea of task-driven computing [9]. In task-driven computing, the computing environment keeps an explicit list of active tasks, i.e. tasks that the user is currently working on or plans to work on. Tasks can be very simple (e.g. talk to a person) or more involved (e.g. teach a class on a regular basis). This task list captures "user intent": it provides the computing environment with information on what the user is trying to achieve.

The Proactive Task Assistant is implemented as a service that periodically checks the user's task list to look for opportunities to help the user. In the current prototype, the task is list is derived from the user's calendar and to-do list, information that is readily available; we assume the user will use specific keywords to label tasks that are candidates for proactive help. The current system provides help with two simple tasks. First, when a meeting is coming up on a specific topic, the Proactive Task Assistant automatically downloads a set of files related to the topic on the user's PDA or laptop so

the user will have them available at the meeting. The user has to specify what files are relevant, for example by keeping the file in a specific folder or set of folders. Second, when the user wants to talk to a person, e.g. Joe, the Proactive Task Assistant periodically checks whether the user and Joe are or will soon be in the same physical area. If that is the case, it will remind the user to talk to Joe. Location information is obtained from the CIS.

4. Experience

In this section we report on our experience in using the CIS to implement four initial implementations of these four applications and in porting the CIS and the applications to a different environment.

4.1. Using the CIS

Our experience using the integrated CIS has been a positive one. The use of a single interface that is familiar to many users (SQL) simplifies the application development and the SQL-like interface appears to be rich enough for the "basic" contextual information that we currently use. Being able to simply request context information instead of having to collect it from diverse sources significantly simplifies application development. Another interesting consequence of using a familiar, powerful interface (SQL) is that it invites users to issue complex queries. For example, the Nearest Printer Service can in principle be implemented as a single SQL query. This is exactly what we were hoping to achieve: simplify application development by moving functionality into the service infrastructure. Unfortunately, our current CIS query synthesizer is not powerful enough to deal with such complex queries, since the diversity of the information providers creates some unique challenges [3]. As a result, our Nearest Printer Service uses two CIS calls: one to the People Location Service and one to an integrated Space and Device Service.

While increasing the set of information provider classes to support the applications, we discovered that the CIS implementation is not quite as simple and clean as Figure 2 suggests. Most providers (e.g. Space Service) are in fact implemented as multiple services, for example a database for static information and one more separate services that collect information on-demand. This raises the question on how to best structure the CIS, e.g. should be in fact implement a single Space Service (that is internally more complex) or should we expose the presence of multiple space-related services to the CIS query synthesizer?

Finally, our experience using a Semantic Web front-end to the CIS shows that such a module can play an important role in supporting rich sets of user preferences and service invocation rules that relate to a variety of contextual attributes and ontologies [10, 11]. We also plan explore using Semantic Web technology to deal with richer contextual information than currently supported by the CIS.

4.2. Porting the CIS

In Fall 2003, we worked with research from III on porting the CIS and applications to their environment. Our experience was that, once the CIS was working, the applications ported very easily. Since this was one of our goals, this was a very encouraging result.

However, we also discovered that porting the CIS was quite time consuming. One time sink was the typical problems associated with porting a new system (e.g. software version mismatches, hardcoded information, ..). These problems will be resolved over time. A more fundamental problem is that

the CIS contains a lot of detailed information and entering this information is tedious and error-prone. Moreover, we should expect that different environments will want to use different information sources (e.g. different calendar systems, badge devices, ..), so porting the CIS may involve building wrappers to integrate these sources the CIS. More research is needed to automate the collection of CIS information, for example by getting the data from existing databases, and to simplify the development of new information providers.

5. Conclusions

In this paper we reported on out experience in using a shared Contextual Information Service (CIS) to collect context information for a set of context-aware applications. We found that the availability of the CIS significantly simplified application development and that semantic web technology allow us to maintain structured context models. However, further research is needed to improve portability and extensibility of the CIS.

Acknowledgements

This research was funding in part by the Institute for Information Industry, by the NSF under award number CCR-0205266, by the Air Force Research Laboratory under contract F30602-02-2-0035 and, earlier, by the Defense Advanced Research Project Agency under contracts N66001-99-2-8918 and F30602-98-2-0135. The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1] Fabien Gandon and Norman Sadeh. Semantic Web Technologies to Reconcile Privacy and Context Awareness. *Web Semantic Journal*, 1(3), 2004.

[2] David Garlan, Daniel Siewiorek, Asim Smailagic, and Peter Steenkiste. Project Aura: Towards Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31, April-June 2002.

[3] Xia Geng. A query optimizer for a distributed contextual information service, December 2003. Masters Thesis, Information Networking Institute, Carnegie Mellon University.

[4] Urs Hengartner and Peter Steenkiste. Access Control to Information in Pervasive Computing Environments. In *Proceedings of 9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, Lihue, Hawaii, May 2003. Usenix.

[5] Urs Hengartner and Peter Steenkiste. Protecting People Location Information. In *First International Conference on Security in Pervasive Computing (SPC 2003)*. Springer Verlag, March 2003.

[6] Changhao Jiang and Peter Steenkiste. A Hybrid Location Model with a Computable Location Identifier for Ubiquitous C omputing. In *The Fourth International Conference on Ubiquitous Computing (UBICOMP 2002)*, pages 246–263, Goteborg, Sweden, September 2002. Springer Verlag. Lecture Notes in Computer Science #2498.

[7] Glenn Judd and Peter Steenkiste. Providing Contextual Information to Ubiquitous Computing Applications. Technical Report CMU-CS-02-154, Department of Computer Science, Carnegie Mellon University, July 2002.

[8] Glenn Judd and Peter Steenkiste. Providing Contextual Information to Pervasive Computing Applications. In *IEEE International Conference on Pervasive Computing (PERCOM)*. IEEE, March 2003.

[9] Joao Pedro Sousa and David Garlan. Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments. In Jan Bosch, Morven Gentleman, Christine Hofmeister, and Juha Kuusela, editors, *Software Architecture: System Design, Development, and Maintenance (Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture)*, pages 29–43. Kluwer Academic Publishers, 2002.

[10] W3C. OWL Web Ontology Language Reference. Working Draft, 31 March 2003, http://www.w3.org/TR/owl-ref/, March 2003.

[11] W3C. RDF Vocabulary Description Language 1.0: RDF Schema. Working Draft, 23 January (2003) http://www.w3.org/TR/rdf-schema/, January 2003.