Issues for Context Services for Pervasive Computing

Maria R. Ebling, Guerney D. H. Hunt and Hui Lei
IBM T. J. Watson Research Center
{ebling,gdhh,hlei}@us.ibm.com

Abstract

Context-aware computing has the potential to greatly alleviate the human attention bottleneck. To facilitate the development of context-aware applications, we envision a context service that provides standardized support for applications. It supports both synchronous queries and asynchronous event notifications. It also addresses privacy concerns by providing controls that allow people to limit the release of their context information.

Introduction

Just as the words or passage that surround a discourse can throw light on the meaning of that discourse, the environment in which a computation takes place can throw light on its meaning or on the intentions and needs of the user that computation is supporting. We call the environment in which a computation takes place its *context*. We define context in the broadest possible sense. A context may either refer to aspects of the physical world or to conditions and activities in the virtual world. Further, context information can be either transient or persistent. Transient context reflects the environment at a single point of time and persistent context is a history of transient context.

Context awareness enables a new class of applications in pervasive computing. These applications can help users navigate unfamiliar territory, find nearby restaurants, receive messages in the most useful and least intrusive manner, get extra sleep when meetings are cancelled, find people with similar interests, and the like. The use of context information in these applications reduces the amount of human attention an application needs to service the user's requests. Because human attention is the ultimate bottleneck in computing, context-aware applications offer an opportunity to increase human productivity in substantial ways.

Context information, by its nature, can contain very private and personal data. For context applications and services to gain widespread acceptance and achieve their goals of reducing the need for human attention, they must address the privacy issues involved and offer protection for the users they monitor.

To date, there has been very little infrastructural support for context-aware applications. Existing applications have been written in an ad hoc manner and are limited in the kinds of context that they use. In fact, most existing applications consider only location awareness. New applications are hard to develop, with significant programming effort spent on deriving and managing context data. In addition, little to no attention has been paid to giving users control over the release of their own context information.

We believe that making context services an integral part of the pervasive computing infrastructure can substantially advance the current state of the art. Applications could then interact with context services to obtain the information they desire without worrying about the details of context management. Further, the costs associated with introducing new context sources can be amortized across many applications.

We are developing and experimenting with a context service, dubbed *Owl*, that supports context-aware applications. Its extensible and flexible architecture accommodates heterogeneous context sources. It offers a programming model that allows for both synchronous queries and asynchronous event notifications. It protects

people's privacy though the use of a role-based access control (RBAC) mechanism.

Uses of Context

Context information may refer either to aspects of the physical world or to conditions and activities in the virtual world. It should support personalized access by delivering the right information to the right place at the right time. It should enable personalized content to be delivered to a smart TV and intelligent sorting of incoming messages. It should allow systems to infer a user's intention and thus to reduce user distraction. For example, a query for restaurants should consider factors such as location as well as food preferences and current wait time of the candidate restaurants. Because pervasive computing devices are, in general, resource constrained, context awareness should enable applications to extend the capabilities of these devices as well as their precious resources by using those available in the local environment. Context awareness should allow users to take advantage of unexpected opportunities. For example, a user could be informed that they will be passing a gas station with prices significantly less expensive than their usual station. Finally, context awareness will allow information to be prefetched in anticipation of future contexts, providing better performance and availability. For example, because applications can gain access to calendar and location information, they can prefetch maps, restaurant information, and local weather/traffic conditions before they are actually needed.

Terminology

Previous work defines the notion of sensed and derived context [Gray2001,Gellersen2000] as well as the idea of physical and logical sensors [Gellersen2000]. Our definition of context includes these ideas, but broadens them to include information beyond the present point in time. Our reasoning is based upon the observation that humans are creatures of habit. Therefore, knowledge of past context information may allow us to infer present (or future) behavior in the absence of current information. Thus, our context service can maintain a history of context information or can use historical data maintained by a sensor.

For example, the system might detect that the receiver located in my office is currently receiving the signal from my active badge. From this information, it might derive that my present location is my office. From an extended history of such information, the system might also derive that I am usually in my office at 9:00am on weekdays.

Owl collects and maintains context information from a variety of context *sources* about numerous *subjects*. *Clients* request context information about one or more *subjects*. Subjects of Owl may be users or objects (e.g., equipment or packages). Clients of Owl may be other users (e.g., a secretary, colleague, or spouse) or other programs (e.g., a service or application). Clients may query for the current context information or may submit a request to be notified when a particular condition is met. The c*ontroller* of context information may be the subject about whom the information pertains or it may be the owner of the object about which the information pertains.

Design Issues

Several considerations influenced our design. Among these were the need to preserve the subject's *privacy* to the greatest extent possible, the need to *scale* to a potentially large number of subjects and clients, and the need to *extend* the system to new sources of context information. These and other factors we considered are discussed in detail in the sections that follow.

Privacy

Any service that maintains context information knows much about its subject. Our instincts impressed upon us the need to protect the privacy of context subjects to the greatest possible extent. People's reaction [Coy1992] to previous research in this area as well as emerging standards [P3P] to protect privacy reinforced these views.

Our privacy protection mechanism is based upon Role Based Access Control (RBAC)

[Sandhu1996]. We assume a closed system: that the identity of all context clients is known to the system. We have chosen RBAC because studies [Ferraiolo1993] have shown that RBAC reduces the cost of administering security policies. RBAC separates the association between users and groups from the associations between groups and privileges. Since the number of groups is typically much smaller than the number of users, RBAC reduces the number of associations that must be managed in most cases and hence the administrative cost. In addition, because subject-based policies align closely with existing business practices and can be expressed naturally in terms of roles, RBAC makes the specification of security policies less error-prone.

A system such as Owl must consider the privacy of context sources as well as that of the context subjects. The identity of context sources should not be released without their explicit permission. This raises the issue of what happens when the privacy of the context subject and that of the source conflict. In such instances, we propose to honor the privacy policy of individuals first and that of the context source second.

Another aspect of privacy concerns whether or not subjects should have control over which context sources can supply data about them. Our opinion on this matter is that the system should at least support such controls.

Scalability

We envision Owl as becoming a piece of the pervasive computing infrastructure. As such, we expect it will need to scale to very large numbers of users. Our initial goal is to support on the order of 10 million subjects with as many as 1 million clients active at any given point in time. When the system grows beyond the confines of a single machine, Network Dispatcher [Hunt1998] or similar load-balancing technology can be used to scale the service. The use of Network Dispatcher assumes that the back-end services are functionally identical. Depending upon the request stream, this assumption may defeat any benefits of caching.

Our desire to support persistent context requires a context store. We have chosen a database system for this purpose. Consequently, the scalability of our system will be limited to the degree that the underlying database system can be scaled.

Extensibility

Because context-aware computing is relatively new to the computing world, services supporting a general notion of context must be easily extensible to accommodate new and unanticipated sources of context information. This requires that the system handle different interaction mechanisms with context sources and that it allow new context sources to be easily added.

A context service can interact with context sources in one of two ways: push or pull. In a *push* mechanism, context sources periodically push updated context information to the context service. The context service maintains the information in a context store and services client inquiries from its local store. In a *pull* mechanism, the context service must explicitly request context information. It can either make this request on a periodic basis (polling) or when an application demand arises. Each mechanism has advantages and disadvantages:

- A push model collects data ahead of need and thus may offer better performance. However, it may consume substantial resources transferring and storing information that is never required. In addition, it must trade off information freshness with the costs of frequent updates.

- A pull system may use fewer resources by obtaining only the data that is required. However, this exposes the context service to inevitable network delays and unavailability. In some circumstances, it may be possible for prefetching and/or caching to alleviate these problems, but this may increase resource utilization.

In both cases, the ability to derive context information from past history is limited by the frequency with which context information is

acquired or by the support offered by the context source.

Another aspect of extensibility is that of accommodating irregularities in context data. One potential solution is to encode the context information in XML. The expressive power and flexibility of XML make it a good representation language for heterogeneous context data. Further, an information model that describes the interrelationships between different types of context data could provide structure for accessing and reasoning about context.

Synchrony

We conjecture that both synchronous and asynchronous operations will be useful. A *synchronous* operation requires that the application wait for a response, whereas an *asynchronous* one notifies the application when the requested information is available. Applications that base their real-time operation on the present context (e.g., deliver this message to the telephone nearest Andy) will require synchronous operations whereas applications that need to be activated upon a particular context (e.g., deliver this message when Andy is free) will find asynchronous operation more useful. Asynchronous operation allows applications to obtain the information they need without resorting to expensive polling behaviors. Allowing asynchronous context queries requires that incoming context changes be matched with outstanding context callback requests. Only experience will show whether the benefits obtained will outweigh the costs imposed.

Quality of Information

Context information often involves real world entities. Thus, it makes sense to measure the quality of context information (QoI), or the extent to which the data corresponds to the real world. The quality of context information can vary, perhaps substantially, depending on the context source. A context service must therefore allow for inaccuracies and uncertainty. Our system allows context information to be associated with QoI metrics such as freshness and confidence. It also allows applications to specify their QoI requirements when interacting with the context service. We should note that the quality of context information, as reported by context sources, remains suspect. Independent monitoring is still needed, as discussed under future work.

Mechanisms for modifying context data or recording complaints must be developed. Currently, social mechanisms must step in should a context source prove to be untrustworthy. Additional work is required to address the issues of monitoring the accuracy of reported QoI and of reporting the quality of context information derived from multiple sources or from historical data.

Early Experiences

We have chosen an incremental, application-driven approach to developing our context service. We have started with a basic system and plan to evolve it gradually, adding new features as required by applications. Currently, our context service provides information on user locations, activities, and on-line presence. We have done an initial usability study on our privacy control interface and expect to refine our interface based upon what we have learned.

In addition to developing the context service itself, we have also developed an application called the Universal Notification Dispatcher (UND). The UND addresses the issue of personal mobility in pervasive computing. A mobile user often has multiple devices. As he moves from place to place, he may switch from one device to another. Like other work in this area [Maniatis1999, Raman2000], the UND delivers messages to users via whatever device (telephone, email, pager, instant messaging, WAP, and SMS) is most appropriate. In contrast to these other efforts, our service relies on the recipient's context information to determine which device is most appropriate. UND in combination with Owl has been used in Planet Blue (a research testbed for pervasive technologies here at IBM) as well as other research projects.

Related Work

An important area of work is that of further supporting applications by providing semantic interpretations of context information. Salber and his colleagues approached this problem by building a context toolkit to aid the developers of context-aware applications [Salber1999]. This toolkit provides widgets that hide the complexity of the actual sensors used to collect the data, abstract this information, and provide reusable and customizable building blocks. A complete infrastructure for context-aware applications requires this type of abstraction. From our recent discussions with Abowd, Dey, and Salber, we believe that their context toolkit could be built using our infrastructure for the raw context information.

The Cricket Location-Support System flips the responsibility for location awareness around [Priyantha2000], providing support for mobile devices to determine their own location within a building. Cricket explicitly does not use location tracking purportedly to "address" the privacy problem inherent in such approaches. By doing so, however, Cricket precludes certain types of applications *unless* an application on the mobile device makes its location known externally. The existence of such an application lands Cricket back at square one with respect to privacy. In contrast, Owl addresses the privacy question head on by giving users control over who can access their context information. Further, Owl supports a *general* notion of context, not just location.

Hull and his colleagues describe a system that, like Owl, is intended to handle many diverse sources of contextual information, though the only source with which they have experience is location data from their custom Pinger device [Hull1997]. Their system goes beyond Owl in the sense that they provide a higher-level abstraction of the information for use by applications (e.g., PersonSpotted events) whereas Owl leaves this functionality for higher levels of the pervasive computing stack. Finally, because their system is intended to serve just a single individual using a wearable computer, scale and privacy were not considered design goals and are not explicitly addressed.

The TEA and Mediacup projects [Gellersen2000] explore an architecture for obtaining and using context in everyday devices. In contrast to many context-based projects, context information in these projects is sensed on the devices themselves. The Mediacup broadcasts the context it detects. These broadcasts could be monitored and stored in a system such as ours. The TEA phone demonstrates a novel use for context information. It shows how context can be exploited to give users, in this case the caller, more information about the current activities of the person they are trying to reach. Though the TEA phone uses self-sensed context, similar functionality could be provided through the use of our system. Neither of these projects consider the problem of privacy – in both cases, context information is treated as openly available and no restrictions are placed upon its dissemination.

Conclusions

Context awareness is critical to achieving the vision of pervasive computing and to alleviating the human attention bottleneck. To date, the development of context-aware applications has been hampered by the need to develop a custom context infrastructure for each application. Owl removes this obstacle by placing the context functionality in the infrastructure. At the same time, it addresses privacy concerns by giving people the ability to easily control the release of their context information. Just as file systems are a standard infrastructural component of computing systems today, we believe that context services should be made an integral part of the pervasive computing infrastructure of tomorrow.

Acknowledgements

We wish to thank our colleagues who have contributed to the development of our context service, including Vincent Bazinette, Azzari Caillier, Ying Chen, J. Smith Doss, Renee Kovales, Peter Malkin, Jussi Myllymaki, Oliver Ribardiere, John Richards, Gregory Stewart, and Robert Sundstrom. We would also like to thank

John Davis II, William Jerome, and Daniel Salber for many interesting discussions.

Bibliography

[Coy1992] Peter Coy. Big Brother Pinned to you Chest. *Business Week*. August 17, 1992, Number 3279, 38.

[Ferraiolo1993] Ferraiolo, D.F.; Gilbert, D.M.; Lynch, N. An Examination of Federal and Commercial Access Control Policy Needs. In *NIST-NCSC National Computer Security Conference* (Baltimore, MD, Sept. 1993), 107-116.

[Gellersen2000] Gellersen, H.-W.; Schmidt, A.; Beigl, M. Adding Some Smartness to Devices and Everyday Things. In the *Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications* (Monterey, CA, Dec. 2000), ACM, 3-10.

[Gray2001] Gray, P. D.; Salber, D. Modelling and Using Sensed Context Information in the Design of Interactive Applications. To appear in the *Proceedings of the 8th IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'01)* (Toronto, Canada, May 2001).

[Hull1997] Hull, R.; Neaves, P.; Bedford-Roberts, J. Towards Situated Computing. In the *Proceedings of the 1st International Conference on Wearable Computing* (1997), IEEE, 146-153.

[Hunt98] Hunt, G.D.H.; Goldszmidt, G.S.; Kind, R.P.; and Mukherjee, R. "Network Dispatcher: a connection router for scalable Internet services", Computer Networks and ISDN Systems, Vol. 30, 1998, 347-357.

[Maniatis1999] Maniatis, P.; Roussopoulos, M.; Swierk, E.; Lai, K.; Appenzeller, G.; Zhao, X. and Baker, M. The Mobile People Architecture. *ACM Mobile Computing and Communications Review (MC2R)*, July 1999.

[OECD] http://www.oecd.org/

[P3P] Platform for Privacy Preferences (P3P) Project. http://www.w3.org/P3P/

[Priyantha2000] Priyantha, N.; Chakraborty, A; Balakrishnan, H. The Cricket Location-Support System. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking* (Boston, MA, August 2000), ACM, 32-43.

[Raman2000] Raman, B.; Katz, P.; Joseph, A. Universal Inbox: Providing Extensible Personal Mobility and Service Mobility in an Integrated Communication Network. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications* (Monterey, CA, December 2000), IEEE, 95-106.

[Salber1999] Salber, D; Dey, A. K.; Abowd, G. D. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of CHI '99* (Pittsburgh, PA, May 1999), ACM, 434-441.

[Sandhu1996] Sandhu, R.S.; Coyne, E.J.; Feinstein, H.L.; Youman, C.E. Role-based Access Control Models. *IEEE Computer*, Vol. 29, No. 2, Feb. 1996, 38-47.

[Wang2000] Wang, H. J.; Raman, B.; Chuah, C.; Biswas, R.; Gummadi, R.; Hohlt, B.; Hong, X.; Kiciman, E.; Mao, Z.; Shih, J. S.; Subramanian, L.; Zhao, B. Y.; Joseph, A. D.; Katz R. H. ICEBERG: An Internet-core Network Architecture for Integrated Communications. IEEE Personal Communications, Vol. 7 No. 4, Aug. 2000, 10-19.

[Weiser1991] Weiser, Mark. The Computer for the 21st Century. Scientific American. September 1991, 66-75.