The Context Fabric: An Infrastructure for Context-Aware Computing

Jason I. Hong
Group for User Interface Research, Computer Science Division
University of California, Berkeley
Berkeley, CA 94720-1776 USA
jasonh@cs.berkeley.edu

Abstract
Despite many sensor, hardware, networking, and software advances, it is still quite difficult to build effective and reliable context-aware applications. We propose to build a context infrastructure that provides three things to simplify the task of building context-aware applications: a context data store for modeling, storing, and distributing context data; a context specification language for declaratively stating and processing context needs; and protection mechanisms for safeguarding privacy needs.

Keywords
Context awareness, context-aware computing, data models, context specification language, implicit input, privacy

INTRODUCTION
Context – the circumstances in which an event occurs; a setting; to join; to weave (American Heritage Dictionary)

A great deal of effort has gone into the field of context-aware computing over the past few years, building applications that have a greater awareness of the physical and social situations in which they are embedded. From a computational perspective, there are four goals for context-aware computing:

- Increasing the number of input channels into computers
- Pushing towards more implicit acquisition of data
- Creating better models that can take advantage of this increased input
- Using the increased input and improved models in new and useful ways

The problem is that it is still extremely difficult to build these kinds of applications. There are five reasons why these goals have been difficult to achieve. First, the same context data can come from many sources. For example, location information can come from active beacons, GPS, or cell phones. Second, the context data is highly distributed, possibly coming from and being used anywhere, anytime. Third, the data models have generally been application-specific and inflexible, making it difficult to share context data. Fourth, these data models have not addressed security and privacy concerns. Fifth, it is difficult to program applications in an environment that is constantly changing in terms of sensors, services, and context data.

Previous work in context-aware computing has focused on the first point. There have been few inroads into the other issues of distribution, modeling, privacy, and robustness. This work focuses on developing an infrastructure, called the Context Fabric, to address these issues [2]. We are designing three key features to help developers create robust applications:

- A flexible and distributed data store to make it easy to model, store, and disseminate context data
- A context specification language for declaratively stating and processing context needs
- Reasonable and customizable privacy mechanisms to help protect context data about end-users

We give a brief overview of each of these below.

CONTEXT DATA STORE
One existing problem with context is how to represent it in a way that many applications can use. Another existing problem is where to store context data and how to distribute it so that it can be accessed on any device, whenever and wherever it is needed. To address these issues, we are designing a context data store, consisting of a logical context data model and a physical data store.

The *logical context data model* is a way of representing entities such as people, places, and things. The context information itself is represented using four concepts: entities, attributes, relationships, and aggregates. *Entities* are simply people, places, and things. Entities represent the base level of context, on top of which more sophisticated representations can be built. Each entity also has access control associated with it, limiting which applications and which people can access its data. *Attributes* describe some property of an entity. For example, people, places, and things all have names. *Relationships* are special kinds of attributes that point to other entities. For example, a person could currently be in a specific place, and this place could contain several things. *Aggregates* are a way of grouping existing entities, and are one way more sophisticated representations of context can be modeled. For example, an action can be modeled as the person doing the action, the place the action takes place, and the things used. A work group can be modeled as a collection of the people in that group. A room can be modeled as a place and all of the things in that place.

The *physical data store* manages how and where the context data is actually stored. The physical data store distributes the data so that copies of the context data can exist in multiple places. For example, a person's private context information might reside in her Palm Pilot and in her computer at home, while her context information at work might reside in an office computer. This approach makes it easier to scale the system up for large numbers of entities and across wide areas. It also increases robustness to failure by improving the availability of context information. Furthermore, it is more efficient to put the context data close to where it is generated and where it is likely to be used. Lastly, it distributes responsibilities in terms of administration, maintenance, and protection of data.

One advantage of this approach is that it decouples context acquisition from context modeling from context usage. For example, a GPS sensor can send a location update. A location service could take this raw sensor data, process it, put it in the right data format, and update the location attribute for the mobile computer. Later on, multiple applications can request and use that context data. This decoupling makes it easier to update and evolve the infrastructure, as well as making it more robust to individual failures.

CONTEXT SPECIFICATION LANGUAGE

The *Context Specification Language* (CSL) addresses the problem of programming context-aware applications in a dynamically changing and heterogeneous environment, with context data distributed across many devices. These issues make it difficult to program context needs procedurally.

CSL is a declarative way of stating context needs at a high level, providing a clean programming abstraction to the context data in the same way SQL does for relational databases. At a conceptual level, CSL statements are things like "What are the five nearest movie theaters to me?" and "Is Scott busy now?" CSL statements are processed by a local Context Service, with one Context Service per device. The Context Service processes CSL statements, locally if all of the context data is available, or expanding and asking other Context Services in the infrastructure if needed. The Context Service handles queries, such as "How many people are in the room right now?", as well as events, such as "Notify me every time a person enters the room."

PRIVACY MECHANISMS

Privacy is perhaps the most debated issue with respect to ubiquitous, context-aware computing. The difficulty here is to find the right balance between the needs of individuals and the needs of governments and societies to properly function. We are currently planning on implementing privacy mechanisms directly into the infrastructure, including restricting certain kinds of context queries to be processed if and only if the person making the query is physically nearby, automatically garbage collecting or aggregating old context data, and allowing context queries to return intentionally ambiguous answers.

EVALUATION PLAN

There are five distinct dimensions for evaluations. The first is to see if the data model is expressive enough to model enough domains that are interesting and useful. The second is to learn if the Context Specification Language is powerful enough to abstract out the complex details. The third is to ensure that the overall system is robust to changes in environment as well as to failures. The fourth is to discover if we have enough useful mechanisms for privacy. The fifth is to find out if the infrastructure makes it easier to develop context-aware applications.

The basic method is to use an iterative design process, working out designs, implementing, deploying, building applications on top, and then refining before going to the next iteration. For the first iteration, we plan to start simple, re-implementing existing context-aware applications and informally evaluating them along the dimensions described above. For the second iteration, we plan to build two more applications, as well as convince others to try building some applications, and see what works and what does not.

RELATED WORK

Schilit's ParcTab system [4] was the first context-aware system infrastructure. We are extending his work by increasing the distributed nature of the context data, adding security and privacy features, and providing a higher level programming interface.

The Interactive Workspaces EventHeap [3] also has similarities. It is a central space where devices and services can post data and events. The EventHeap is designed for connecting devices in a local room together. We are taking some of the ideas from the EventHeap and seeing if we can adapt them for context-aware applications.

The Context Toolkit [1] is closest to our work. The Context Toolkit took an "operating systems" approach, providing an abstract layer for sensors. Hardware is primary, while data formats and modeling are secondary. In contrast, we take a "database" approach, focusing more on how the data will be modeled, distributed, protected, and used, making data primary, and hardware secondary.

ACKNOWLEDGEMENTS

Thanks to my advisor James Landay, Mark Newman, Keith Edwards, and Anind Dey for comments. Thanks also to DARPA, Intel, and Xerox PARC for funding this research.

REFERENCES

[1] Dey, A.K., D. Salber, and G.D. Abowd, A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction (HCI) Journal*, 2001. **16**(2-3).

[2] Hong, J.I. and J.A. Landay, An Infrastructure Approach to Context-Aware Computing. *Human-Computer Interaction (HCI) Journal*, 2001. **16**(2-3).

[3] Johanson, B., A. Fox, P. Hanrahan, and T. Winograd, The Event Heap: An Enabling Infrastructure for Interactive Workspaces. 2000. http://graphics.stanford.edu/papers/eheap

[4] Schilit, B.N., *A Context-Aware System Architecture for Mobile Distributed Computing*, 1995. http://www.fxpal.xerox.com/people/schilit/schilit-thesis.pdf