<Context-Aware> schema </Context-Aware>

Brian Meyers and Amanda Kern
Microsoft Corp., One Microsoft Way, Redmond WA 98052 USA
{brianme, amandak} @ Microsoft.com
http://www.research.microsoft.com/research/easyliving/

Abstract

Context for the computer can be decomposed into three parts – information about the computing environment, information about the physical environment, and information about the user's history. In order to author a context-aware application, this information must be available in a standardized fashion.

The EasyLiving Project at Microsoft Research is building a prototype smart environment that supports several context-aware applications. The system maintains a number of data stores that encapsulates information about the context that these applications require. For example, the system uses computer vision to sense the location of people and stores this information, along with other geometric measurements, in a geometric model of the physical world. The system also has data stores regarding the user's history and preferences and the current state of the computing environment.

The Movie Player application uses information about the physical location of the person and the person's expressed preference to automatically start the desired movie whenever the user sits on one of the couches. The video output is routed to the correct screen based on the user's location. When the user leaves the area, the movie is stopped automatically.

The "Warmer/Colder" game hides a virtual object in the physical room. It then uses the physical location of the user to provide feedback on their relative location to the object. For example, "You are getting warmer," or "Getting colder."

The AnyWhere Mouse is a radio frequency mouse that uses the location of the user and the location of computers in the physical world to direct its input to the nearest computer. This allows the user to carry the mouse with him and use it on nearby computers.

Even though the applications mentioned here are relatively simple, the additional context that they understand and exploit makes user characterize them as 'smart'. This leads to several issues. What information should be stored? Where should it be stored? Who is allowed to retrieve it? When is it available? Why should this be standardized? And finally, how should it be stored?

What information should be stored?

The more context shared between the two parties in an interaction the more efficient that interaction becomes. The user has their entire history of interactions as context. In addition, the environment around them affects them: what objects are available, whether it is too noisy to hear system sounds, or if nearby activity is distracting. In order to facilitate the interaction with the user, the computer must be able to describe and understand the user's context. Since context builds over time it is critical to store not just the current data but also a history of information that can be searched for trends and changes.

Context for the computer can be decomposed into three parts – information about the computing environment, information about the physical environment, and information about the user's history (where history implies all the preferences they have selected). A context-aware application should be able to dynamically discover this information. Building a system that can store all contextual information is untenable, but it is necessary to design a system that allows an application to express what contextual data it requires and have that system extended to accommodate that requirement.

Where should it be stored?

Any user of multiple computer systems recognizes that information about their preferences is not being shared effectively. Ironically, the World Wide Web increased the motivation for collecting context about the user while at the same time reducing the amount of information sharing. Not only is this

information not shared between sites, it is often stored on a particular machine resulting in even more user frustration.

Despite the ever-increasing trend to being online all the time, the reality is that networks will never have 100% continuous connectivity. Information that lives only on a single machine is subject to both hardware and network failures. Server farms have the required reliability but mobile devices will frequently be out of contact with the server[1]. In order to provide the most continuous access to the data, it must be stored redundantly and cached locally.

Who is allowed to retrieve information?

Contextual information can unfortunately be used both to aid and to harm the user. For example, a malicious person could exploit location information to physically harm the user. Security for information access is critical. Each request for information must be accompanied by a description of the originating entity. Generated data will have to be given an access control list based.

When is information available?

Although some applications only need to query the current state, the real advantage comes from events generated by contextual changes. Recall the earlier examples of the video pausing when the user leaves the room. This requires the application have the ability to articulate the events that it needs, and a service to distribute those events. Current eventing systems expose only those events that the authors deem important; however, a context-aware application must be able to request notification based on any relevant change in the observed context. In order to reduce overhead, this request should provide filtering that could specify maximum frequency and other criteria for the notification.

Why should the information be in a database?

Today information about the computing environment is typically provided using function calls. Modern systems provide API like *GetSystemColors* or *getpeername*. The difficulty with this approach is three-fold. First, as we continue to add more and more information the number of functions becomes unwieldy. Second, the applications themselves provide context that is not being exposed. The "current text selection" is an example of context that is interesting but difficult to obtain in a robust fashion. Third, using functions provides no mechanism for notification. Applications that want to respond to changes in the information must track those changes themselves. The current trend of using APIs to expose context information is approaching its scaling limit.

The full potential of context-aware applications is not realized until they can access the state of the operating system, the hardware, nearby devices, other running applications as well as having access to machines that the user previously interacted with. In fact, this requires that the context-aware application expose its state as part of the computing context. In addition to the enormous number of API this would require, every application or process that generated contextual information would have to be running and reachable in order to answer questions about its state. To overcome these problems, everything that generates contextual data should write that data into a shared data store.

How should the information be stored?

Modern databases have several desirable properties, including the ability to back up data, partially replicate data and locally cache data. However, even after the contextual information is placed into well-designed, robust, and secure database, there is still the issue of discovery. It must be possible to examine the schema and locate the applicable data. By adopting a self-describing standard, like XML, and requiring that meta-information be stored with the data, this schema can be examined and transformation can be applied[2].

Efforts are underway to facilitate common schemas and standardize transforms for some limited domains. While in many arenas this will be sufficient, a more robust approach is to use a natural language interpreter to read the schema and meta-information. The results of that analysis can be used to understand the data and its structure or to locate the appropriate transformation utilities.

---

[1] If for no other reason than to conserve battery life.

[2] While an extreme example, the recent loss of the Mars Probe was caused by a lack of meta-information.

Summary

The EasyLiving project provides three contextual data services – Persona for user histories, GeoModel for physical world information, and Foreman for digital world information. It is likely that as the amount of contextual information grows these services will become further specialized.

As the contextual information changes, each sensor (where sensor could be either an application reporting its current state or an actual sensor like the vision system) contacts the appropriate service and updates the information. Applications can query the current state of the data or subscribe to change notifications.

Any program writing information into the context service is required to provide meta-information that describes the data. This description can be read by the application and if necessary, the appropriate transformers can be used to access it.

As EasyLiving progresses the amount of available context information will continue to grow. Context-Aware application development should proceed without knowledge of how the information was collected or from where. In a fully self-describing system, context-aware applications should even be able to accommodate new information without the author having to alter the code.

References

[coen98] Coen, M. et al. "Design Principles for Intelligent Environments", AAAI Spring Symposium on Intelligent Environments, March 23-25 1998, AAAI Press TR SS-98-02.

[dey99] Dey, A. et al. "A Context-Based Infrastructure for Smart Environments", Managing Interactions in Smart Environments, Springer-Verlag, 1999, pp. 201-213.

[Holtman99] Holtman, K. "Transparent Content Negotiation in HTTP", http://gewis.win.tue.nl/~koen/conneg/rfc2295.html

[schmidt99] Schmidt, A et al. "Advanced Interaction in Context", Handheld and Ubiquitous Computing, Springer-Verlag, 1999, pp. 89-101.

[shafer98] Shafer, S. et al. "The New EasyLiving Project at Microsoft Research", Proc. DARPA/NIST Smart Spaces Workshop, July 1998, pp. 127-130.

[W3C99] "Platform for Privacy Preferences (P3P) Project", http://www.w3.org/P3P/

[W3C99] "XML Schema", http://www.w3.org/TR/xmlschema-1/