A Classification Framework for Storage and Retrieval of Context

B.I.J. Siljee, I.E. Bosloper, J.A.G. Nijhuis

University of Groningen, Department of Computing Science
PO Box 800, 9700 AV Groningen, The Netherlands
{b.i.j.siljee, i.e.bosloper, j.a.g.nijhuis}@cs.rug.nl

Abstract. An increasingly important requirement for mobile and pervasive systems is the capability to adapt at runtime to a changing context; they must be context-aware. Much current work in this field focuses on application-specific solutions and ad-hoc approaches, and the lack of conceptual models for the design of context-aware systems hinders the development of more general and complex systems. In this paper we present a classification framework for the storage and retrieval of context, which supports the development of context-aware systems. We demonstrate the usefulness of the framework by modeling a number of context-aware systems.

1. Introduction

Small networked devices like Personal Digital Assistants (PDAs) and mobile phones are rapidly becoming more common in every day living. They are cheaper, smaller, more powerful and more ubiquitous than before. The fast changing environment of these devices – combined with the increasing need for more intelligent and autonomous behavior – requires software that can adapt to changes in its environment. This is what we call *context-aware* systems: systems that can keep track of their context and change according to context changes.

The specific context of a system depends on its application domain. Schilit [1] divides context into three categories:

- *Computing context*: available CPU time and memory, network connectivity, communication costs and bandwidth, nearby resources such as printers, displays and workstations.
- *User context*: user's profile, user's location, and people nearby.
- *Physical context:* lighting, noise levels, traffic conditions, and temperature.

Chen [2] adds a fourth category:

- *Time context:* the time of day, and the day, week, month, and season of the year.

A controlled and simple way to implement system changes is to stop, update and restart the entire system. However, this is not always desirable or even possible. Multiple reasons for modifying a system while it remains running are:

- *Mobile:* Access to a system may be restricted, for instance when systems are 'in the field'. In this case systems cannot be updated through offline methods.
- *Critical:* Mission-critical systems need to remain running; they cannot be stopped and restarted.
- *Extensible:* Some systems are capable of integrating third-party software at runtime. This allows for functionality not available at release or startup time. An example is internet browsers that must integrate a plug-in to display a movie in-page.
- *Comfort:* Changing a system without restarting does not block users, and therefore increases usability.

We define systems that change at runtime, i.e. without stopping, as *dynamic systems*.

To further increase usability, effort is focused on the ability of a system to self-detect when and what to change, and to make this change autonomously. These self-adaptive systems measure the degree in which they satisfy their requirements runtime; the variation in these measurements is usually due to a changing context.

Based on the above, we conclude that automatic dynamic systems, i.e. systems that automatically change at runtime, need to be context-aware. Furthermore, the environment of mobile devices changes rapidly. Not only the environment of such a device itself changes, but mobile devices may also move out of the environment they were originally designed for, thus facing new environments. Dynamic, context-aware systems should be able to adapt runtime to these unforeseen context changes as well.

A software architecture provides a global perspective on the software system in question. Architecture-based design of software systems provides major benefits [3], as it shifts the focus away from implementation details towards a more abstract level. However, most research on context-aware systems focuses on ad-hoc solutions at the implementation level, as designers lack abstractions to reason about context-aware systems [4] [5]. This hinders the reuse of design paradigms and ideas.

The contribution of this paper is an architectural classification framework of the abstractions that designers can use for the development of context storage and retrieval in an application-independent fashion. The framework gives an overview and description of the different possibilities of context storage and retrieval, thereby providing designers with the possible choices and consequently simplifying design decisions. We show the usefulness of this framework with two cases in which we design context-aware systems.

The remainder of the paper is organized as follows. In the next section we present the classification framework for context storage and retrieval. Subsequently, we present the two cases, followed by related work. Finally, the paper is concluded.

2. Context Storage and Retrieval Classification Framework

The concept of context-awareness in systems can be divided into three separate phases (see also Figure 1):

1. Monitoring of the environment
2. Interpretation of the monitoring data through the context model
3. Adaptation of the system to a changed context

Probes monitor the environment and output raw data. This monitoring data needs to be interpreted before it has a meaning useful to the system; without this meaning the monitoring data rarely holds the context information the system needs. The interpretation is usually done through a *context model*, which translates the monitoring data into *context information*. The context information is then used to reconfigure the system. This division is common for most application-specific context-aware systems, for example in GPS car navigation systems (e.g. Hertz [6] and Navman [7]), in mobile handheld devices enhanced with sensors (e.g. the stick-e notes [5]), or in power saving applications (e.g. [8]).

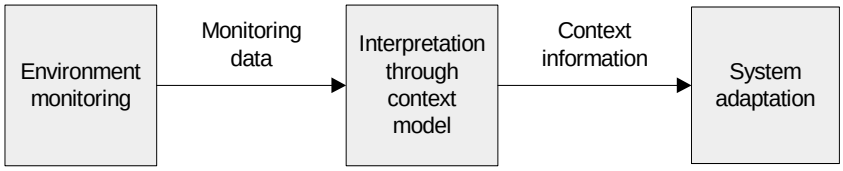

Fig.1: Three phases in context-aware systems. The monitoring of the environment by probes results in raw monitoring data, which is interpreted through a context model. The resulting context information is used by the system to base system changes on.

We present a classification framework for the storage of the context model and the retrieval of the context information from the context model. For the monitoring of the environment an excellent framework is given by Dey [4]. For the adaptation of the system to changed context we refer to [9], as this falls outside the scope if this paper.

Context model storage

The context model is used to add context meaning to the raw monitoring data, and is thus a necessary element for most context-aware systems. Several options exist for the storage of the context model.

Degree of distribution:

- *Central*: the context model is stored at one central location, e.g. as one database as in [10], or as one Hidden Markov Model as in [11]. In order to retrieve the current context information, the monitoring data from one or more probes is combined with the centrally stored context model.
- *Distributed:* the context model is stored at multiple sites, which can be networked locations. We distinguish two cases:
  1. Each location has an identical context model. This is useful for fault-tolerance and speed of context information retrieval: as the context model can be located near or inside the different systems that need it, the systems can retrieve their context information faster.
  2. Each location has a different part of the context model, and the context models from several locations must be combined for the complete context model.

An example is CoolAgent of HP [12], where the context model is distributed among different agents.

Location relative to the system:

- *Extrinsic*: the context model is stored at a different location than the system itself; it is a service for context information. This way, systems are dependent on the service to retrieve their context information. Many location-based context-aware systems use an extrinsic service, as, for instance, a traffic jam information service.

  Such extrinsic context information services are useful when the context model is space-consuming and it is not possible to locate the context model within the system, for example in small, handheld devices. Extrinsic context model storage is also useful when the context model needs to be updated transactional and all the different devices must have access to the new context information at the same time. However, systems are dependent on the service for their context information, which introduces an extra reliability risk.
- *Intrinsic*: the context model is stored inside the system itself. This is useful for small context models, if the context model is different for each system, or they may even be necessary if the systems cannot always reach a context service. An example of the latter may occur in a battle situation, in which it is likely that such a context service will at some point become unavailable due to attacks.

Context information retrieval

After the context model adds meaning to the raw monitoring data, the resulting context information is used by the system. Different aspects in the process of retrieving the context information from the context model are as follows:

Initiative:

- *Context-push*: the system receives its context information without needing to send a request first every time it is needed. This is typical for, for instance, GPS car navigation systems.
- *Context-pull*: the system explicitly requests the current context information, for example, through service requests when the context model is extrinsic, or via a function call when the context model is intrinsic.

Timing:

- *Event-driven:* systems only get their context information when some event occurs. This is usually the case when context information is discrete. An example of this is a context change caused by a person walking into a different room, or a mobile phone moving into a different cell. A non-discrete example is of a running program that has the available system memory as its context. When a second program is started that uses a lot of system memory, less memory is available for the running program. This is the event on which it responds by using less memory-intensive, but slower, versions of its components.
- *Periodic:* at certain scheduled points in time the context information is retrieved or received. This is suitable for continuous context information.

History:

- *Absolute*: context information is taken at absolute points; previous context information is not used. GPS navigation systems, for instance, only consider the current location.
- *Relative:* the difference of the context information over time is important. Therefore the context information history must be stored and must be possible to retrieve. This may be more space and time consuming.

Information presentation:

- *Explicit:* The raw monitoring data *is* the context information; no context model is needed for interpretation. The raw data presented to the system explicitly states the context information and the system directly uses this data as-it-is.
- *Implicit:* The system infers the context information from its input data by using a context model.

Dynamism of Context Models

Not always every possible environment of a system can be foreseen at system development time. The environment of a route planner for example includes highways. When new highways are built after deployment of the route planner, the route planner does not show the shorter route that is now possible. These unforeseen changes can result in a context model that no longer interprets the monitoring data accurately. Different degrees of dynamic adaptation capabilities of a context model are:

- *Static:* the context model never changes. This means that the same monitoring data always results in the same context information.
- *Updatable:* it is possible to update the context model.
- *Dynamic:* the environment changes over time, and the context model continuously adapts to these changes at runtime, i.e. it is dynamic. For instance, context models consisting of adaptive neural networks keep on learning during operation.

Figure 2 shows a schematic overview of the framework.

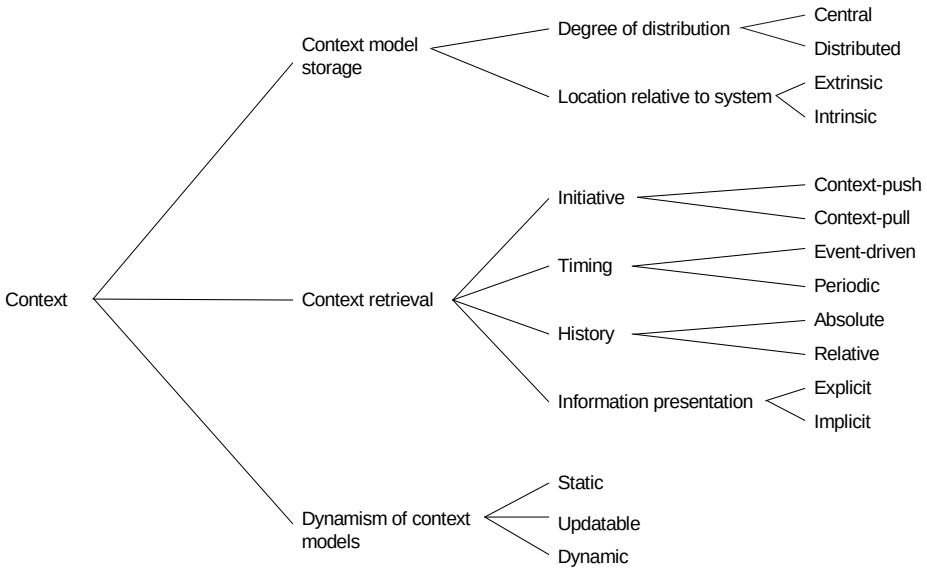

Fig. 2: A classification framework for the modeling and retrieval of context.

3. Context-aware systems

In this section we demonstrate how to use the classification framework presented above. We describe two cases of context-aware systems, and for each case we give the corresponding architecture in terms of the framework.

3.1 Tourist Navigation System

Situation

A local government wants to rent handheld devices with city maps to tourists, enabling tourists to find their way to the city's tourist attractions. The handheld device contains a map of the city, an accurate location sensor, and a digital compass. The tourist simply selects the desired destination, and the handheld device displays directions to get there from wherever the tourist is located, keeping the directions accurate while the tourist is moving.

Furthermore, the city is expanding rapidly, attractions can be closed, walking routes blocked; this would all result in continuously inaccurate directions from the handheld devices. Therefore it would be desirable to update the handheld devices regularly.

Architecture

The architecture of such a system is given in Figure 3. The system consists of a probe that continuously feeds sensor data to a route component. The route component

contains a map component that holds the context model: the city map. The context information is a combination of the current location and the navigation instructions, and is continuously sent to the viewer component. Retrieval of the system's context can therefore be classified as a periodic context-push. The viewer component displays the route on the display of the handheld device. The context model is stored only in the map component, which lies inside the system, making it a central and intrinsic context model. The context information results implicitly from the monitoring data, as the tourist's current location and the route to the destination are based on the city map.

To update the context model, only the map component needs to be updated. The probe, the route calculating program and the viewer do not need to be changed. Explicitly locating the context model as the city map simplifies the updating process.

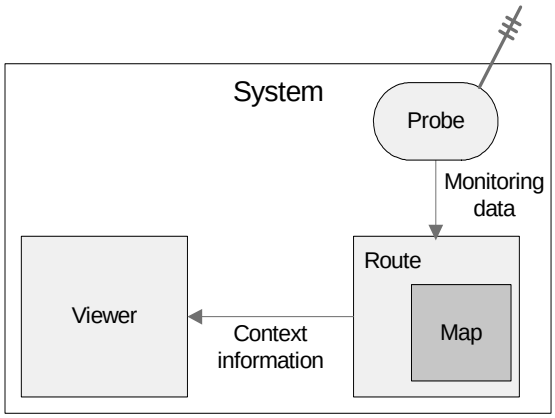

Fig. 3: Architecture of a context-aware, updatable, navigation system.

3.2 Museum Tour Guide

Situation

A museum offers its visitors small handheld devices that automatically display information about the exhibits exposed in the room the visitor walks around in ([4] [13] [14]). These devices are very small and do not contain much capacity for data storage, so they cannot store all the information about all the exhibits present in the museum. Furthermore, the museum frequently changes the location of the exhibits among the different rooms, adding new exhibits and removing others. Therefore each exhibit provides information about itself upon request.

Architecture

The system architecture is shown in Figure 4. Each exhibit presents its information explicitly to the handheld device when the latter enters the room. The handheld devices emit a signal, and upon reception of this signal by the exhibit the context information is sent back. The information about the exhibits is directly displayed on the device by the viewer component.

The context information is stored externally: not in the handheld device, but with the exhibits. The context information is also distributed over the different exhibits, as

each exhibit only contains information about itself. After a request for information (the signal from the device) the exhibits explicitly present their information; no context model is needed to interpret the exhibit information. Furthermore, this system performs a context-pull based on the event of entering a room. As the context information is explicitly presented by the exhibits, no context model is needed. Therefore nothing needs to be adapted when an exhibit is moved from one room to the other. When a visitor with a handheld enters the new room, the context information from the exhibit is automatically sent again.

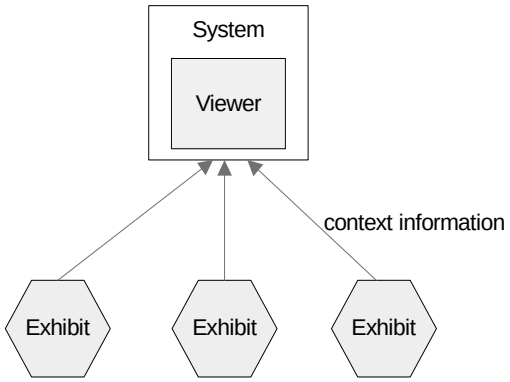

Fig. 4: Architecture of a context-aware museum tour guide.

4. Related Work

In 1994 Schilit [1] describes four classes of context-aware applications along two dimensions: whether the task at hand is getting information or doing a command, and whether it is effected manually or automatically.

Pascoe [15] proposes a taxonomy of features for context-aware systems, which includes contextual sensing, contextual adaptation, contextual resource discovery and contextual augmentation (the ability to associate digital data with a user's context). This taxonomy divides the process of retrieving context in several steps, but does not give an overview of the options for each step.

Similarly, Dey [4] also presents a taxonomy of features that a context-aware system might support: presentation of information and services to a user, automatic execution of a service, and tagging of context to information for later retrieval.

Chen [2] divides context-aware systems into two categories: an *active* context-aware system automatically adapts to discovered context by changing the system's behavior; a *passive* context-aware system presents the new or updated context to an interested user or makes the context persistent for the user to retrieve later. Chen furthermore distinguishes between a symbolic location model (representing location as abstract models) and a geometric location model (representing location as coordinates). Different data structures for the expression and exchanging of context infor-

mation include key-value pairs, tagged encoding, object-oriented models, and logic-based models. Finally, Chen divides possible context-aware architectures in centralized and distributed.

Sun [16] gives three main methods to obtain context information: explicit query, polling, and event-driven.

However, most research on context-awareness has focused on creating reference architectures for specific application domains. Examples of these architectures are the Context Toolkit [4], the Stick-e Notes [5], and HP's CoolAgent [12]. Most of these architectures contain elements for the storage and the retrieval of context information, and often the implementation of the elements differ between different architectures. These differences served as a basis for the classification framework presented in this paper.

5. Conclusions

Context-awareness is a necessary feature of automatic dynamic systems that need to adapt to context changes. Many current context-aware systems are application-specific, and are often developed in an ad-hoc fashion. However, architectural abstractions are needed to build large and complex context-aware systems, because not explicitly modeling context-awareness can result in unnecessary complex implementations. We divided the concept of context-awareness into three phases: 1) environment monitoring, 2) interpretation of the monitoring data through the context model, and 3) adaptation of the system to a changed context. Architectural abstractions exist for the first and for the third phase, see for example Dey [4] for the first phase and Oreizy [9] for the third phase. However, to our knowledge no architectural abstractions exist for phase two: the interpretation of monitoring data through a context model. In this paper we presented these abstractions in a classification framework for the storage of the context model and the retrieval of the context information. This framework supports the development of context-aware systems by offering an overview of the possible choices a designer has. We demonstrated the use of the classification framework by modeling two context-aware systems.

6. Acknowledgements

This research has been sponsored by ConIPF (Configuration in Industrial Product Families), under contract no. IST-2001-34438. The ConIPF project aims to define and validate methodologies for product derivation that are practicable in industrial applications.

References

[1] B. Schilit, N. Adams, R. Want, "Context-aware computing applications", *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, pp. 85-90, Santa Cruz, California, December, 1994, IEEE Computer Society Press.

[2] G. Chen, D. Kotz, "A Survey of Context-Aware Mobile Computing Research", Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, November, 2000.

[3] M. Shaw, D. Garlan, *Software Architecture: Perspectives on an emerging discipline*, Prentice Hall, 1996.

[4] A.K. Dey, D. Salber, G.D. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications", *Anchor article of a special issue on context-aware computing in the Human-Computer Interaction (HCI) Journal*, Volume 16 (2-4), pp. 97-166, 2001.

[5] P.J. Brown, J.D. Bovey, X. Chen, "Context-Aware Applications: from the Laboratory to the Marketplace", *IEEE Personal Communications*, Volume 4 (5), pp. 58-64, 1997.

[6] Hertz, NeverLost, Web page available at: http://hertzneverlost.com.

[7] Navman Europe, Web page available at: http://www.navman-europe.com.

[8] O. Cakmakci, J. Coutaz, K. Van Laerhoven, H. Gellersen, "Context Awareness in Systems with Limited Resources", *Artificial Intelligence in Mobile Systems (AIMS)*, ECAI 2002, pp. 21-29, 2002.

[9] P. Oreizy, M. Gorlick, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, A. Wolf, "An Architecture-Based Approach to Self-Adaptive Software," *IEEE Intelligent Systems*, Volume 14 (3), pp. 54-62, May/June, 1999.

[10] A. Harter, A. Hopper, P. Steggles, A. Ward, P. Webster, "The Anatomy of a Context-Aware Application", *Wireless Networks*, Vol. 8 (2-3), pp. 187-197, February, 2002.

[11] T. Yonezawa, B. Clarkson, M. Yasumura, K. Mase, "Context-aware Sensor-doll as a Music Expression Device", *CHI2001 Proceedings Extended Abstracts*, pp. 307-308, ACM SIGCHI, 2001.

[12] H. Chen, S. Tolia, "Steps towards creating a context-aware agent system", TR-HPL-2001-231, HP Labs, 2001.

[13] B.B. Bederson, "Audio augmented reality: A prototype automated tour guide", *Proceedings of the CHI'95 Conference on Human Factors in Computing Systems*, pp. 210-211, New York, NY, ACM Press, 1995.

[14] S. Fels et al., "Progress of C-MAP: A context-aware mobile assistant", *Proceedings of AAAI 1998 Spring Symposium on Intelligent Environments*, Technical Report SS-98-02, pp. 60-67, March, 1998.

[15] J. Pascoe, "Adding Generic Contextual Capabilities to Wearable Computers", *Proceedings of the 2nd International Symposium on Wearable Computers*, pp.92-99, Pittsburgh, Pennsylvania, October, 1998, IEEE Computer Science Press.

[16] J. Sun, J. Sauvola, "Towards a Conceptual Model for Context-Aware Adaptive Services", *4th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp. 90-94, Chengdu, China, August, 2003.