Modelling and Using Imperfect Context Information*

Karen Henricksen
CRC for Enterprise Distributed Systems Technology and
School of Information Technology and Electrical Engineering, The University of Queensland
kmh@dstc.edu.au

Jadwiga Indulska
School of Information Technology and Electrical Engineering, The University of Queensland
jaga@itee.uq.edu.au

Abstract

Most recently developed context-aware software applications make unrealistic assumptions about the quality of the available context information, which can lead to inappropriate actions by the application and frustration on the part of the user. In this paper, we explore the problem of imperfect context information and some of its causes, and propose a novel approach for modelling incomplete and inaccurate information. Additionally, we present a discussion of our experiences in developing a context-aware communication application, highlighting design issues that are pertinent when developing applications that rely on imperfect context information.

1. Motivation

Pervasive computing presents a novel set of design challenges that demand radically new software engineering techniques. Context-awareness is often touted as a solution that meets these challenges by enabling software applications to exhibit the required levels of flexibility and autonomy. Context-aware applications exploit information about the context of use, such as the location, tasks and preferences of the user, in order to adapt their behaviour in response to changing operating environments and user requirements. This information is gathered from sensors or from human users.

Context-aware applications typically assume that the context information upon which they rely is complete and accurate. However, this assumption is usually unjustified, as sensed context information is often inaccurate or unavailable as a result of noise or sensor failures, while user-supplied information is subject to problems such as human error and staleness.

Consequently, usability problems arising from reliance on imperfect context information are sometimes observed in context-aware applications. For example, Benford et al. recently presented an interesting discussion of the implications of using imperfect location data in an mixed-reality game [1]. They noted that errors in location information often led to confusion when game players were observed to jump around unpredictably in the virtual environment, but were also exploited by sophisticated users for tactical advantage.

Clearly, context-aware applications must be developed with an understanding of the problems inherent in gathering reliable context information, and also of the attendant design issues. In this paper, we explore these challenges. We characterise various types and sources of imperfect context information, present a set of novel context modelling constructs that accommodate these, outline a software infrastructure that supports the management and use of imperfect context information, and describe our experiences with using the context modelling approach and infrastructure.

2. Characterising context information

2.1. Types of imperfection

In this section, we characterise four types of imperfect context information. We consider imperfection with respect to specific properties or attributes. These are aspects of the context that can be described by atomic pieces of information, such as the location or

* The work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Industry, Science & Resources).

Type	*Source*	*Persistence*	*Quality issues*	*Sources of inaccuracy*
Sensed	Physical and logical sensors	Low	May be inaccurate, unknown or stale	Sensor errors and failures; network disconnections; delays introduced by distribution and interpretation
Static	User/administrator	Forever	Usually none	Human error
Profiled	User (directly or through applications)	Moderate	Prone to staleness, may be unknown	Omission of user to update in response to changes
Derived	Other context information	Variable	As for base types + subject to further errors introduced by the derivation process	Imperfect inputs; use of a crude or oversimplified derivation mechanism

Table 1. Typical properties of context information.

activity of a given user, or a capability of a computing device. A property is:

- *unknown* when no information about the property is available;
- *ambiguous* when several different reports about the property are available (for example, when two distinct location readings for a given person are supplied by separate positioning devices);
- *imprecise* when the reported state is a correct yet inexact approximation of the true state (for example, when a person's location is known to be within a limited region, but the position within this region cannot be pinpointed to the required (application-determined) degree of precision); or
- *erroneous* when there is a mismatch between the actual and reported states of the property.

Various combinations of these four classes of imperfection are possible (e.g., amongst a set of ambiguous reports, at least some will be imprecise or erroneous).

Unknowns usually result from connectively problems and sensor and other failures. Ambiguous information arises when the value of a property can be derived independently from multiple sources. Where possible, ambiguous information should be resolved early using conflict resolution techniques. However, this is not always possible, implying that ambiguous information must sometimes be exposed to, and meaningfully exploited by, context-aware applications. Imprecision is common in sensor-derived information, while erroneous context information arises as a result of human error and the use of brittle heuristics to derive high-level information from crude inputs.

2.2. Classes of context information

We partition context information into four classes as shown in Table 1. Specifically, we recognise three principal sources of context information: sensors, human users and derivation from other types of information. *Sensed* context information is usually frequently changing, and suffers from problems such as inaccuracy and staleness. User-supplied context information is usually by necessity slowly changing, as it is unreasonable to expect users to keep frequently changing information current. This type of information is either *static* (never changing and therefore highly accurate) or dynamic. We term the latter type of information *profiled* context. This is usually obtained directly from users in the form of user profiles, or indirectly from the user's applications (e.g., from scheduling software that maintains a history of user activities). Profiled information is often out-of-date or incomplete. Finally, the characteristics of *derived* information are usually largely determined by those of the base information types, however further errors and inaccuracies can be introduced by the derivation process.

3. Related work

Few of the software infrastructures and modelling abstractions developed in the field of context-awareness address the problem of imperfect context information. Here, we briefly survey some of the exceptions.

Schmidt et al. propose a layered software architecture for abstracting context information from sensors, which models context as a vector of pairs [9]. Each pair consists of a value and an accompanying probability estimate. The programming primitives provided alongside the architecture allow actions to be associated with the detection of contexts with predefined levels of probability. Judd and Steenkiste describe a generic interface for querying context services that allows clients to specify their quality requirements as bounds on accuracy, confidence, update time and sample interval [7]. Context services endeavour to meet the requirements, and explicitly signal failures to do so in query responses. Finally, Lei et al. present a context service that accepts freshness and confidence metadata from context sources, and passes this along to clients so that they

can adjust their level of trust accordingly [8]. The service also allows clients to express desired quality levels in queries.

These solutions have several shortcomings which we address in our work. First, as discussed in an earlier paper [5], almost all currently used context modelling approaches lack the required formality and expressiveness to capture rich types of context information and support reasoning about context. Further, even the solutions that accommodate some types of imperfect information neglect to adequately accommodate unknown and ambiguous information. Lei et al. acknowledge the problem of ambiguity, but leave it to clients of their context service to resolve it.

4. Modelling context information

We developed our own context modelling approach designed to overcome these shortcomings based on Object Role Modeling (ORM) [2]. In this section, we introduce ORM and discuss the use of our context modelling extensions to describe imperfect information and to differentiate the four types of context information characterised in Section 2.

Our modelling approach is designed to support a variety of tasks across the software lifecycle. First, its graphical notation is well suited for use by the developer in the task of identifying and specifying the context requirements of an application, and exploring related information quality issues. The mapping of the modelling concepts to a relational data model (described in an earlier paper [6]) supports a straightforward translation of the graphical model to a context management system (implemented as an extended and possibly distributed database) that aggregates, stores and responds to queries on context information at runtime. Finally, further context modelling and programming abstractions that are closely integrated with the modelling approach presented here (described in a companion paper [4]) serve to simplify the task of implementing flexible context-aware applications.

4.1. Basic modelling constructs

ORM's basic modelling construct is the fact type. The modelling process chiefly involves identifying the required fact types and annotating these to indicate constraints on populations. The instantiation of an ORM model consists of a set of facts that conform to the specified constraints. Fact types are drawn in ORM's notation as a series of role boxes, where each box is attached by a line to an object type that participates in the role. Each fact type is annotated with a name and one or more uniqueness constraints (similar to key constraints in the relational model), indicated by double-headed arrows over a subset of the roles. Object types are drawn as ellipses containing a name and an optional reference mode in parentheses that describes the representation of instances (e.g. by identifier or name). An example model is shown in Figure 1. The annotations on the fact types principally represent our extensions to ORM and are discussed in the following sections.

4.2. Modelling classes of context information

The differentiation of the four classes of context information described in Section 2 is important for context management purposes (as, for example, sensed context information requires different treatment to static information in terms of conflict detection and update management). Derived fact types are already supported by ORM; these are shown by attaching an asterisk to the relevant fact type and supplying a rule that describes the production of derived facts from the base fact type(s). In order to distinguish the remaining types of information, we introduce an annotation scheme that explicitly marks each non-derived fact type as sensed, static or profiled, as shown in Figure 1.

In our example model, locations of people and computing devices are acquired from sensors. Proximity of users to devices is inferred from these two sensed types. Information about permissions of people to use devices is supplied by a human (e.g. an administrator). Finally, device type information is static.

4.3. Modelling alternatives

When a fact type can contain ambiguous or conflicting information it is marked as an alternative type using the '*a*' annotation shown in Figure 1. Each alternative type has a special alternative uniqueness constraint that spans all but one role of the fact type. Facts appearing in an instance of the fact type are viewed as alternatives when they have identical values for all of the roles spanned by this uniqueness constraint.

In the model of Figure 1, the locations of people are represented by an alternative fact type. This implies that each person can have several recorded locations (corresponding to separate sensor readings).

The difference in semantics between alternative and ordinary facts must be taken into account when reasoning about context. Our approach is to apply a three-valued logic in which ordinary facts are viewed as truthful and alternative facts as possibly true [4, 3].

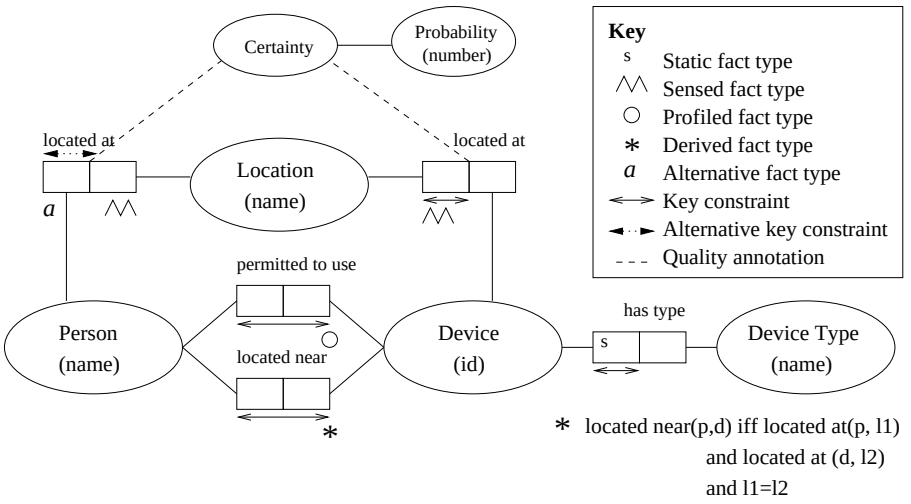

Figure 1. Example context model.

4.4. Modelling information quality

We also extend ORM to allow facts to be associated with relevant quality indicators that allow the end users of the information (usually context-aware applications) to make judgements about the level of confidence they invest in it. Each fact is associated with zero or more quality parameters, which in turn are characterised by one or more concrete metrics. The example model shown in Figure 1 associates facts belonging to the two sensed types with a single certainty measure, expressed as a probability estimate.

4.5. Representing unknown context

In order to support effective reasoning about context information, it is important to distinguish between false and unknown information. We capture unknowns using *null* values, following the approach commonly used in database systems. Facts that are not present are assumed false under a closed world assumption [4].

4.6. Other modelling constructs

We provide further extensions to ORM that allow the modelling of historical information and dependencies between different types of context information. These fall outside the scope of this paper, but are described elsewhere [6, 3, 5].

5. Management and use of context information

We have implemented software support for our context modelling approach in the form of an infrastructure that manages context information from a variety of sources and facilitates the exploitation of this information by applications. A high-level view of the main components of this infrastructure is shown in Figure 2.

The responsibilities of the infrastructure are threefold. First, the context gathering components (sensors, interpreters and aggregators) acquire and process sensed context information. Second, a distributed context management system assumes the role for integrating, storing and managing context information from a range of sources, responding to context queries, and generating notifications of significant context changes. The context management system also provides interfaces for users to easily update and browse their static and profiled context information. A set of plug-in components, which we term context receptors, perform fact-type-specific management tasks. In the case of sensed context information, these are responsible for mapping the heterogeneous context information produced by the context gathering components into the fact abstraction used by the context management system, routing queries from the management system to the appropriate components, and detecting and resolving conflicts and ambiguity where possible. Finally, a programming toolkit allows context-aware applications to easily exploit the services of the context management system using the programming models and abstractions described in a companion paper [4].

6. Experiences and discussion

As a case study, we developed a context-aware communication application using the context modelling approach and software infrastructure described in the pre-

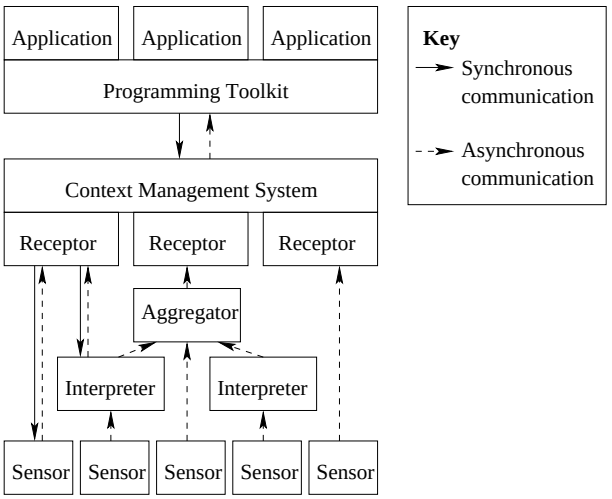

Figure 2. Architecture of the context-awareness infrastructure.

vious sections. We now present a brief discussion of our experiences in this case study, focusing on design issues related to the use of imperfect context information.

The application provides agent-based support for context-sensitive choice of communication channels for interactions between users. Each agent manages the interactions of a single person using relevant preference and context information as described in [4].

The context model used by the application is an extension of the model shown in Figure 1. It covers associations between people and their communication devices and channels, locations of people and devices, relationships between people, and historical information about user activities. Most of the information is profiled or static, and highly reliable. The exceptions are user location information, which is often ambiguous, and activity information, which is often unknown.

The application addresses the problem of imperfect information in part by retaining the user in the decision loop. This allows inappropriate choices, arising from reliance on flawed context information or preferences that imperfectly reflect user requirements, to be detected and overriden. Additionally, preference information is exposed to users so that it can be corrected or evolved if necessary. However, we found that when the preference set was large or complex, it was not always easy for users to identify (and then change) the preference(s) responsible for producing undesirable choices. In the future, we would like to add a mechanism that allows users to flag poor choices as they arise, and to view and manipulate the specific preference and context information that led to a given choice.

7. Concluding remarks

The problem of imperfect context information represents a significant obstacle to the success of context-aware applications, yet is commonly overlooked. In this paper, we attempted to partially remedy this situation by presenting a novel approach to modelling and using imperfect context information. We also discussed experiences gained through a case study, highlighting a small sample of the challenges and design issues that we encountered. Further information about the case study can be found in [4].

References

[1] S. Benford, R. Anastasi, M. Flintham, A. Drozd, A. Crabtree, C. Greenhalgh, N. Tandavanitj, M. Adams, and J. Row-Farr. Coping with uncertainty in a location-based game. *IEEE Pervasive Computing*, 2(3):34–41, July-September 2003.

[2] T. A. Halpin. *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*. Morgan Kaufman, San Francisco, 2001.

[3] K. Henricksen. *A framework for context-aware pervasive computing applications*. PhD thesis, School of Information Technology and Electrical Engineering, The University of Queensland, Submitted September 2003.

[4] K. Henricksen and J. Indulska. A software engineering framework for context-aware pervasive computing. In *2nd IEEE Conference on Pervasive Computing and Communications (PerCom)*, Orlando, March 2004.

[5] K. Henricksen, J. Indulska, and A. Rakotonirainy. Modeling context information in pervasive computing systems. In *1st International Conference on Pervasive Computing (Pervasive)*, volume 2414 of *Lecture Notes in Computer Science*, pages 167–180. Springer, 2002.

[6] K. Henricksen, J. Indulska, and A. Rakotonirainy. Generating context management infrastructure from context models. In *4th International Conference on Mobile Data Management (MDM) - Industrial Track*, Melbourne, January 2003.

[7] G. Judd and P. Steenkiste. Providing contextual information to pervasive computing applications. In *1st IEEE Conference on Pervasive Computing and Communications (PerCom)*, pages 133–142, Fort Worth, March 2003.

[8] H. Lei, D. M. Sow, J. S. Davis, G. Banavar, and M. R. Ebling. The design and applications of a context service. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):45–55, October 2002.

[9] A. Schmidt and K. Van Laerhoven. How to build smart appliances. *IEEE Personal Communications, Special Issue on Pervasive Computing*, 8(4):66–71, August 2001.