Generating Context Management Infrastructure from High-Level Context Models*

Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy

School of Information Technology and Electrical Engineering
The University of Queensland
and Distributed Systems Technology Centre
{karen, jaga, andry}@itee.uq.edu.au

Abstract. The trend towards pervasive computing is driving a need for applications that are highly autonomous, as well as responsive to changes in their environments and in user requirements. Such applications are commonly termed context-aware, and are reliant on accurate information about aspects of their surroundings such as the location, goals and activities of users. Unfortunately, the construction of context-aware applications is a difficult task, and, for this reason, various infrastructures supporting the construction of such applications by assisting with the gathering, processing and dissemination of context information have recently appeared [1–3]. However, these are based on models of context that lack formality and expressiveness and offer little support for the design tasks associated with context-aware applications. In this paper, we present a context modelling approach that offers a means for developers to describe and program with context at a high level, without the need to consider issues related to context gathering, management or representation. We also describe a mapping process that transforms high-level context models to management systems capable of maintaining and supplying context information to applications at run-time.

1 Modelling context information

The context modelling approach that we describe in this paper came about as a result of two observations. Firstly, the models of context currently in use in context-aware systems typically lack formality and expressiveness, and are primarily concerned with representing context in a form suitable for applications to query; this means that they are of little use to the application designer as a basis for exploring and specifying the types of context information required by a context-aware application. We described these problems in detail in [4], and motivated the need for a more expressive modelling approach on the basis of our experiences in using CC/PP to build context-aware systems. In addition, commonly used conceptual modelling approaches, such as UML and ER, are not well suited to capturing special features of context information, such as:

* The work reported in this paper has been funded in part by the Co-operative Research Centre Program through the Department of Industry, Science and Tourism of the Commonwealth Government of Australia.

- Histories of information
- Uncertain and incomplete information
- A range of different classes of information with diverse characteristics, including sensor-derived and user-supplied information
- Dependencies between different types of information

While modelling approaches exist that are able to capture some of these aspects (such as temporal ER modelling languages [5] and quality modelling approaches [6]), we are not aware of any single approach that can capture all of these features in a natural way.

In our initial attempt to define a set of modelling concepts appropriate for describing context information, we avoided aligning our work with any of the existing data modelling approaches. This gave us the flexibility to explore the most natural way of capturing the aforementioned aspects of context. The results are described in [7]. Subsequently, we have decided to reformulate our modelling concepts as extensions to the Object-Role Modeling (ORM) approach [8]. The choice of ORM over alternative approaches was largely motivated by the suitability of ORM's fact types as an abstraction upon which concepts such as histories and quality annotations can be built, and their close alignment to the association abstraction upon which the original model was based. The benefits of using ORM include the ability to express a variety of constraints that are important in (but not specific to) the context modelling problem, and the ability to leverage its straightforward mapping to the widely used relational model.

In the remainder of this paper, we characterise ORM and present a set of extensions we have made to ORM to accommodate the special characteristics of context information described above. We also outline the issues involved in mapping our context modelling concepts to a context management system built upon a relational database.

Core concepts in ORM The basic modelling concept in ORM is the fact; consequently, the modelling of a domain using ORM involves identifying appropriate fact types and the roles that entity types play in these. Diagrammatically, entity types are represented as ellipses containing a name and an optional reference scheme that describes the representation of instances of the entity type. Entity types that are simple value types do not require a reference scheme and are shown as dashed ellipses. Fact types are shown as sequences of role boxes, with each box attached to an entity type. Each fact type has a label that describes its semantics in natural language. Bars (or double-headed arrows) are placed over roles in a fact type to indicate which roles or combinations of roles must not contain duplicate values. These notations are illustrated in the example of Figure 1. ORM also supports a variety of advanced constraint types, such as mandatory roles and value and subtype constraints. For further information on these, the reader is referred to [8].

Capturing different classes of context. We extend ORM to allow fact types to be categorised according to their persistence and source. Broadly, we classify

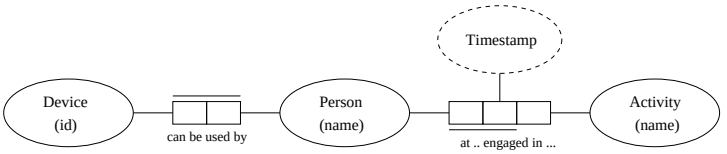


Fig. 1. Modelling a binary and ternary fact type using ORM

fact types as either static or dynamic. Static fact types are populated by facts that remain unchanged as long as the entities they describe persist; an example might be a "Device d is of type t" fact type, because, once true, facts of this type remain true thereafter. When defining a static fact type, the set of roles against which facts are invariant must be declared by annotating these roles with an s, as shown in the diagram at the left of Figure 2 (a); we refer to this set of roles as the base set. By marking a fact type in this way, two constraints are implied: i) fact instances cannot be modified; and ii) fact instances can only be deleted when an entity participating in one of the roles of its base set is deleted.

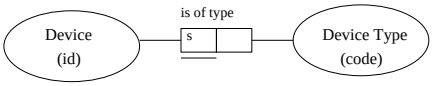
Dynamic fact types can be classed as profiled, sensed and derived types, depending on the source of the facts with which they are populated. Profiled facts are supplied by users, whereas sensed facts are supplied by hardware or software sensors and derived facts are obtained from one or more facts using a derivation function. The notations we introduce to distinguish the three classes of dynamic fact type are shown in the diagrams at the left of Figure 2, (b)-(d).

Capturing histories. Histories can be captured in ORM by involving timestamps in fact types (as in the ternary fact type of Figure 1). However, the frequency with which histories occur in context modelling prompted us to introduce a special temporal fact type to capture time-indexed data. This is shown in Figure 2 (e). The annotation of a fact type with a temporal constraint is similar to (but semantically richer than) including timestamps, representing start and end times, as roles that participate in all uniqueness constraints of the fact type.

Capturing fact dependencies. Fact dependencies represent a special type of relationship that exists between facts. A fact $f1$ is considered to depend upon a second fact $f2$ if and only if a change in the state of $f2$ can lead to a change in the state of $f1$. A state change may involve a change to one or more role values or deletion of a fact altogether. The existence of such dependencies is captured by the binary, transitive *dependsOn* relation. We represent dependencies diagrammatically by drawing a dashed arrow from a dependent fact type to the type that it depends upon and annotating this arrow with a predicate that describes which particular instances of the two fact types are linked by the *dependsOn* relation, as shown in Figure 2 (f).

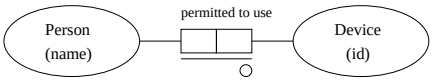
Capturing quality. Dynamic facts can be of varying quality depending upon factors such as their source and age. Our approach to dealing with this problem

(a) Static Fact Type



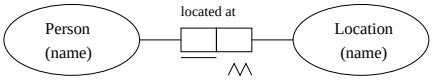
IsOfType(DeviceID, DeviceType)

(b) Profiled Fact Type



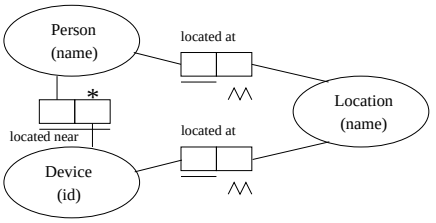
PermittedToUse(PersonName, DeviceID)

(c) Sensed Fact Type



LocatedAt(PersonName, LocationName)

(d) Derived Fact Type



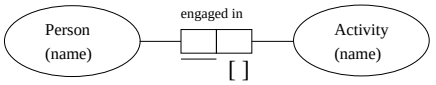
* located near(p,d) iff p located at l1
and d located at l2
and l1=l2

LocatedNear(PersonName, DeviceID)

LocatedNear is a view corresponding to the following SQL:

```
CREATE VIEW LocatedNear AS
SELECT PersonName, DeviceID
FROM PersonLocatedAt, DeviceLocatedAt
WHERE PersonLocatedAt.LocationName =
    DeviceLocatedAt.LocationName
```

(e) Temporal Fact Type



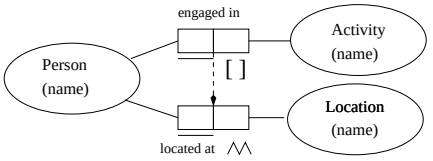
EngagedIn(PersonName, StartTime, EndTime, ActivityName)

EngagedInNow(PersonName, Activity)

EngagedInNow is a view corresponding to the following SQL:

```
CREATE VIEW EngagedInNow AS
SELECT PersonName, ActivityName
FROM EngagedIn
WHERE CURRENT_TIMESTAMP >= StartTime and
    (CURRENT_TIMESTAMP <= EndTime or EndTime is null)
```

(f) Fact Dependency



engaged in(p1,a) dependsOn located at(p2,l) iff p1 = p2

(g) Quality Annotation

LocatedAt(PersonName, LocationName, ProductionTime, StandardError)

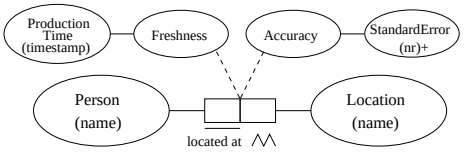


Fig. 2. Modelling examples

is to allow facts to be associated with quality indicators that can help users to decide how much trust to place in the information. The designer of the context model must determine, for each dynamic fact type, the dimensions of quality that are relevant, such as accuracy or certainty (we term these quality parameters), and appropriate ways of concretely measuring the quality of a fact against these dimensions (metrics). Figure 2 (g) shows the annotation of a fact type with two quality parameters (freshness and accuracy), which each possess a single metric.

2 Mapping to a context management system

In mapping from a context model to a context management system, we leverage ORM's relational mapping procedure, Rmap [8]. The benefit of this approach is that the majority of the constraints that can be expressed in ORM can be enforced at run-time by a relational database, while the remaining context management tasks are performed by a lightweight management infrastructure built upon the database. In this section, we describe some mapping and management issues associated with the modelling concepts described in the previous section.

Mapping the context classes. The mapping of static, profiled and sensed context to relations is performed as defined by the Rmap procedure, except that each fact type is always mapped to a unique relation[1]. Examples mappings are shown in Figure 2 (a)-(c), with the relational representation for each fact type shown on the right-hand side.

Derived context types are mapped to views. Figure 2 (d) illustrates how the view corresponding to a simple derived fact type might be defined using SQL.

Additionally, the context management system maintains a catalogue of the classification (static, sensed, derived or profiled) of fact types and the base set of each static fact type. This information is used to manage updates: for example, to ensure that the constraints associated with static fact types are not violated.

Mapping histories. Temporal fact types are mapped as described above, with the addition of two attributes named StartTime and EndTime, both of type Timestamp, which participate in each key of the relation. In addition, a view is defined for each temporal fact type that supports easy querying by capturing only those facts that are applicable at the present time. Figure 2 (e) shows an example mapping. The uniqueness constraint over the role played by *Person* implies that at any given time, a person is engaged in only one activity. This constraint is not fully expressed in the key constraint of the corresponding relation, and must be enforced separately by the context management system. Further, the implicit constraint that StartTime precede EndTime must be enforced.

[1] In contrast, Rmap groups fact types with identical keys into the same table. We have chosen to preserve the fact abstraction in the relational representation to support straightforward logic-based querying in which simple boolean queries (such as LocatedNear("Ann", "Library")) can be easily applied to relations.

Mapping fact dependencies. Fact dependencies have no explicit relational representation, but instead are stored and managed separately by the context management infrastructure. Dependencies are recorded by capturing the two fact types involved, together with a predicate that can be evaluated with respect to a pair of instances of these fact types to determine whether they are related by the *dependsOn* relation. The context management system exploits its knowledge of dependencies to trigger i) updates to dependent facts (for example, by invoking a sensor to refresh a sensed fact) and ii) notifications to event listeners of possible changes to dependent facts, whenever these actions are appropriate.

Mapping quality annotations. The quality annotations of a fact type are mapped into the type's relational representation as additional attributes. An attribute is included for each quality metric. Quality measures are queried and updated in the same manner as the context information they describe. An example mapping is shown in Figure 2 (g).

3 Concluding remarks

Recently proposed context-awareness infrastructures are built around models of context that are implicit and informal, and largely lacking in the expressive power required to accommodate features such as varying information types, quality and temporal characteristics. In this paper, we described a context modelling approach based on ORM that was designed to overcome these shortcomings. This modelling approach is sufficiently abstract to allow application developers to specify and program with context at a high level, while being formal enough to support a straightforward mapping to a context management infrastructure based on the relational model.

References

1. Dey, A., Salber, D., Abowd, G.: A context-based infrastructure for smart environments. In: 1st International Workshop on Managing Interactions in Smart Environments (MANSE'99). (1999)
2. Schmidt, A., et al.: Advanced interaction in context. In: 1st International Symposium on Handheld and Ubiquitous Computing (HUC'99), Karlsruhe (1999)
3. Harter, A., et al.: The anatomy of a context-aware application. In: Mobile Computing and Networking. (1999) 59–68
4. Indulska, J., et al.: Experiences in using CC/PP in context-aware systems. In: Proceedings MDM 2003 (to appear), Melbourne (2003)
5. Gregersen, H., Jensen, C.S.: Temporal entity-relationship models - a survey. IEEE Transactions on Knowledge and Data Engineering **11** (1999)
6. Wang, R., Reddy, M.P., Kon, H.: Towards quality data: An attribute-based approach. Decision Support Systems **13** (1995) 349–372
7. Henricksen, K., Indulska, J., Rakotonirainy, A.: Modeling context information in pervasive computing systems. In: Proceedings of Pervasive 2002, Zurich (2002)
8. Halpin, T.: Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design. Morgan Kaufman (2001)