Modelling Context Information with ORM*

Karen Henricksen[1], Jadwiga Indulska[2], and Ted McFadden[1]

[1] CRC for Enterprise Distributed Systems Technology (DSTC)
karen@itee.uq.edu.au, mcfadden@dstc.edu.au

[2] School of Information Technology and Electrical Engineering,
The University of Queensland
jaga@itee.uq.edu.au

Abstract. Context-aware applications rely on implicit forms of input, such as sensor-derived data, in order to reduce the need for explicit input from users. They are especially relevant for mobile and pervasive computing environments, in which user attention is at a premium. To support the development of context-aware applications, techniques for modelling context information are required. These must address a unique combination of requirements, including the ability to model information supplied by both sensors and people, to represent imperfect information, and to capture context histories. As the field of context-aware computing is relatively new, mature solutions for context modelling do not exist, and researchers rely on information modelling solutions developed for other purposes. In our research, we have been using a variant of Object-Role Modeling (ORM) to model context. In this paper, we reflect on our experiences and outline some research challenges in this area.

1 Introduction to Context Modelling

Context-awareness has recently emerged as a popular approach for building applications for mobile and pervasive computing environments that are capable of automatically adapting to their environments and reducing the need for explicit directions from the user. Context-aware applications monitor and respond to information about the context of use, such as the available computing resources and the current location and activity of the user. Context-aware applications typically obtain this information - which is termed *context information* - from varied sources, including people, sensors (e.g., GPS receivers, tilt sensors and accelerometers), network monitors, and other context-aware applications. Common examples of context-aware applications include tourist guides that present information that is tailored according to the user's location and preferences, and mobile phones that adapt their ringing behaviour depending on where the user currently is, who they are with, and what they are likely to be doing. A variety

* 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 Education, Science, and Training).

of emerging ‘intelligent environments’ - such as homes, hospitals and meeting rooms that are able to sense the activities of their occupants and leverage this information to automatically derive those occupants’ requirements - can also be considered context-aware.

Early context-aware applications, which typically relied on only one or two simple types of sensed context information, were generally constructed by writing software modules to directly query sensors and carry out simple types of interpretation of the sensor outputs. In these applications, the model of context information - or, more specifically, the *kinds* of context used, and their *representations* or *formats* - were fixed and closely intertwined with the application logic. Today, the shortcomings of this approach are widely acknowledged. Increasingly, developers of context-aware applications are seeking techniques for modelling context information in a uniform way, in order to:

– promote sharing of context information and context-sensing infrastructure between applications;
– facilitate complex queries over multiple types of context information;
– decouple the details of the model from the application logic, thereby simplifying the process of evolving the model; and
– provide a common framework for representing both sensor-derived data and information from other sources, both of which are important for context-aware applications.

As yet, there are no well-established techniques for modelling context, as Strang and Linnhoff-Popien [1] demonstrated in their recent survey. Most researchers have adopted modelling approaches from other fields, such as database modelling (ORM, ER) and the Semantic Web (CC/PP, DAML+OIL, OWL). A major advantage of these approaches is that most are widely understood and supported by common tools and query languages. On the other hand, there is often a considerable degree of mismatch between the capabilities of these modelling approaches and the characteristics of the context information that needs to be modelled, which necessitates extensions or work-arounds. It is this problem that we address in this paper, drawing on our experiences with using an ORM-based context modelling approach to develop a variety of context-aware applications. We highlight some requirements for modelling context that are not met by ORM (or other similar modelling approaches) and present a set of novel extensions that we have developed to address some of these requirements. We also show how our extended variant of ORM is used to support the development of context-aware applications, and outline some of the remaining challenges in the area of context modelling. We assume that the reader already has a basic familiarity with ORM; for a comprehensive treatment, we refer the reader to the excellent book by Halpin [2].

2 Requirements for Modelling Context Information

The traditional use of ORM is the modelling of business domains, in order to support the development of databases that are principally populated and used by humans. Our goals in modelling context information are quite different. Context models are mapped to information repositories that are populated by many different entities, including humans, hardware and software sensors and context-aware applications. Context repositories are primarily queried by context-aware applications, which use the information to determine the current situation and requirements of their users. These differences in information production and consumption mean that ORM does not provide the most natural solution for modelling context. In this section, we highlight a set of specific context modelling problems which ORM either does not address well or at all.

2.1 Distinguishing Information from Different Sources

In order to manage and use context information effectively, it is necessary to distinguish between sensed, human-supplied and application-supplied information. Sensed context information is generally updated frequently but can be inaccurate due to problems like noise, calibration/configuration errors, and so on. In comparison, user-supplied information is updated infrequently. It is typically accurate initially, but becomes less reliable over time. Application-supplied information has characteristics that fall between those of sensed and user-supplied information. We require techniques for associating fact types with information sources, and, in some cases, annotating individual facts with specific metadata about their sources (e.g., sensor or user IDs).

2.2 Allowing Inconsistency and Incompleteness

Conflicting information is a common problem in context-aware systems; different sensors, for example, may report different values, and it may not be possible to determine which value is the correct one. Likewise, incomplete information is the norm rather than the exception. ORM uses a variety of constraints, such as uniqueness constraints and mandatory role constraints, to prevent such problems from arising; these are mapped to database constraints, so that updates that violate the constraints are rejected at run-time. This is acceptable when information is inserted by humans who can investigate the cause of the conflicts and resolve them. However, it is not appropriate for context-aware systems. Instead, inconsistencies and incompleteness should be allowed in a controlled fashion, and appropriately handled by context-aware applications. This requires separate modelling constructs to capture true domain constraints, which match the real world semantics, and the looser integrity constraints that should be enforced by context repositories.

2.3 Modelling Temporal Data and Constraints

Context-aware applications are frequently not only interested solely in the current context, but also in future or past states, or changes in state over time. Therefore, it is often desirable to model *histories* of certain types of context information. Although histories can be modelled in ORM by explicitly including time as an additional role in fact types, this solution is clumsy; it is more natural to provide dedicated modelling constructs for temporal fact types, including special temporal constraint types. A considerable amount of work has been done in the area of modelling temporal data, and some solutions do exist for extending ORM with temporal concepts [3]; however, these are not yet part of the standard ORM notation.

2.4 Modelling Information Quality

As discussed in Sections 2.1 and 2.2, context information is often imperfect, and context-aware applications must be capable of operating under this assumption. To allow applications to make informed decisions about which context information should be trusted and which should not, quality metadata is required. Appropriate types of metadata vary between fact types; for example, a fact describing a person's current activity might be associated with a timestamp, while a fact describing a person's current location coordinates might also be annotated with the standard error associated with the sensor supplying the information. Although quality could be modelled simply by including additional roles in fact types, this solution becomes undesirable when it comes to reasoning about context information, as quality values cannot be distinguished from ordinary values in facts.

2.5 Modelling Information Ownership

A final requirement is imposed by privacy requirements in context-aware systems. In business domains, an information repository generally falls under the control of a single business entity, which can set global access control policies for users of the repository; unfortunately, this is not the case for context repositories, which may combine sensitive information belonging to many users [4]. This necessitates a more complex model of ownership and control. One way to address this requirement is to extend the context model with statements that explicitly distribute the ownership of individual information types (i.e., fact types) amongst a set of people and/or other entities.

3 The Context Modelling Language

Although solutions exist for some of the problems discussed in the previous section (for example, for modelling temporal data [5] and information quality [6]), there is no single modelling approach that addresses all of the problems in a cohesive way. In this section, we outline a set of extensions to ORM that we developed to address most of the problems. For want of a better name, we refer to this ORM variant as the Context Modelling Language (CML).

3.1 Source Annotations

Our first extension allows fact types to be characterised in terms of the persistence and source of the information that they capture. The motivation for this extension was provided in Section 2.1. First, we differentiate between *static* and *dynamic* fact types. Static fact types represent invariant properties of a context-aware system. Static facts are very simple to manage; they are usually stored indefinitely in context repositories for querying by context-aware applications.

Dynamic fact types are classified according to source. *Sensed* fact types represent information supplied by hardware or software sensors, while *profiled* fact types represent information supplied by users or context-aware applications. The former are generally frequently updated, while the latter are not; therefore, they suit different styles of management within context repositories. Sensed information that is only infrequently queried may not be kept up to date within context repositories (so as to conserve resources), but instead loaded on demand by querying the appropriate sensors.

The annotations we use to represent static, profiled and sensed fact types are illustrated in Fig. 1 (a)-(c). Note that these annotations are never attached to ORM's derived fact types.

3.2 Alternative Fact Types

In order to deal with conflicts of the kind that we described in Section 2.2, in which sensors report different values for some type of context, we introduce the notion of *alternatives.* Alternatives are mutually exclusive facts (of the same fact type) that have been reported about some entity or entities. Example alternatives are "Michelle is located in Sydney" and "Michelle is located in New York".

To selectively allow alternatives, we introduce an alternative fact type. This is annotated with an 'a' symbol and a special alternative uniqueness constraint, as shown in Fig. 1 (d). The uniqueness constraint always spans n-1 roles, where n is the arity of the fact type. The role *not* spanned by the constraint is known as the *alternative role.* Alternative uniqueness constraints are distinguished from ordinary uniqueness constraints to indicate their distinct semantics. An alternative uniqueness constraint is a domain constraint that is not strictly enforced by context repositories; instead the constraint is effectively extended over the alternative role to allow alternative values for this role. Alternative facts have different semantics to ordinary facts, which needs to be taken into account when querying context repositories. We discuss this issue briefly in Section 3.6.

3.3 Temporal Fact Types

To accommodate histories of context information, we extended ORM with a temporal fact type. This extension uses an entirely different notation to the recent TORM proposal [3], as it pre-dates TORM. The notation is shown in Fig. 1 (e). A fact type is marked as a temporal fact type using the '[ ]' annotation, which has the effect of associating all facts with a *valid time* [7], expressed as an interval having a start time and an end time.

Uniqueness constraints on temporal fact types can be either *snapshot* or *lifetime* constraints, in the terminology of [8]. A single fact type may have both types of uniqueness constraint. CML adopts the convention that all constraints on temporal fact types are, by default, lifetime constraints. This preserves the normal

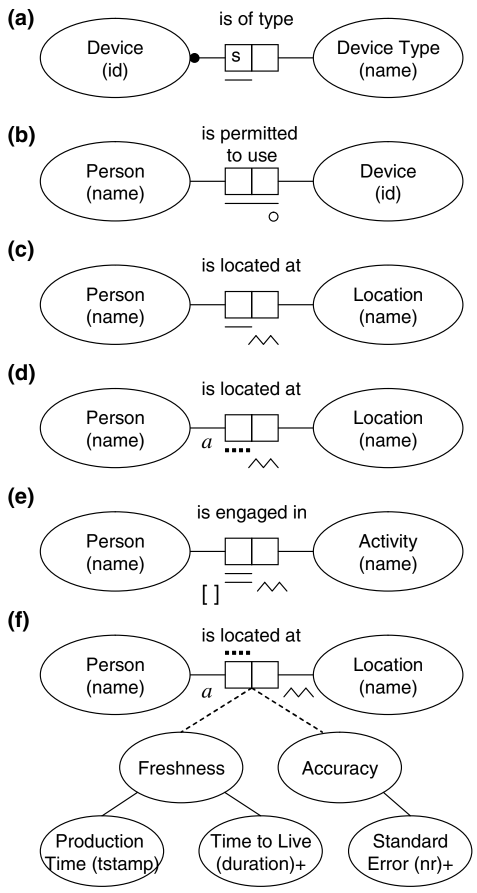

Fig. 1. CML's context modelling extensions. **(a)**, **(b)** and **(c)** show static, profiled and sensed fact types, respectively. **(d)** provides an example of an alternative fact type; this allows multiple location readings to be associated with each person. **(e)** illustrates a temporal fact type used to capture histories of user activities. **(f)** shows an alternative fact type with quality annotations.

semantics of the constraints when the timestamps that are implicit in temporal fact types are regarded as objects participating in ordinary roles. Snapshot uniqueness constraints are drawn using the special notation shown in Fig. 1(e). These express constraints that apply at any given point in time, but not globally over fact histories. For example, the constraint in Fig. 1 (e) indicates that a person has at most one activity at any time (but can have many activities within a history). An external variant is also supported, in which double, rather than single, lines are used to connect the encircled u to the participating roles.

Care must be taken when combining temporal fact types with other kinds of fact types. In general, derived temporal fact types are nonsensical unless at least one of the base fact types involved in the derivation is also temporal. There are also some complications associated with combining temporal and alternative fact types, which are outside our scope here, but are documented in [9].

3.4 Quality Annotations

To support decision making about imperfect context information by context-aware applications, CML provides constructs for annotating fact types with quality annotations that effectively allow quality metadata to be attached to individual facts. CML's quality constructs are partially inspired by work of Wang et al. [6,10] that addressed quality modelling in relation to ER. As illustrated in Fig. 1 (f), each fact type can be annotated with zero or more quality parameters (here, *freshness* and *accuracy*). These parameters are associated with one or more metrics (*production time*, *time to live* and *standard error*), which indicate how the quality is measured/recorded for each fact.

An alternative modelling approach, which does not require special constructs, would be to objectify the fact type to which the quality annotations are attached and add one new binary fact type for each quality metric, in which the objectified fact type plays one role and the metric plays the other. However, our notation is more natural and compact, and can have its own specialised mapping when creating context repositories from CML models.

3.5 Ownership Statements

Finally, to support privacy, we have created a textual notation for specifying the ownership of facts. Ownership statements are specified as part of a *context schema*, which is a textual form of a CML model that we created as a convenient form of input for tools that perform manipulations and mappings on context models. We will discuss these tools in Section 4.

To keep the task of specifying ownership manageable even for large numbers of fact types, we primarily associate ownership with objects rather than facts, and then assign a default ownership to each fact by forming the union of the ownerships of the objects participating in the fact. This default ownership can also be explicitly overridden by providing ownership statements for fact types. A full discussion of the ownership scheme can be found in an earlier paper [4].

3.6 Relational Mapping, Querying and Interpretation

One of the attractions of ORM is its mapping to the relational model, and we have extended this mapping to incorporate our context modelling constructs. It is not possible, owing to space constraints, to describe the details of the mapping procedure in this paper; however [9] provides a full discussion. In Section 4, we will briefly describe a tool that we have developed to automate the procedure.

While there are no *representational* problems associated with mapping our extended ORM to the relational model, there are problems of *interpretation.* As mentioned in Section 3.2, an alternative fact does not have the same semantics as an ordinary fact. To overcome this problem, we provide our own query layer for context repositories that maps portions of context queries to standard relational database queries expressed in SQL. The query layer provides evaluation of context information using a three-valued logic which is able to accommodate alternative facts. We plan to extend the query mechanism to provide more sophisticated treatment of quality annotations, unknowns and temporal facts.

4 Tool Support for Developing Context-Aware Systems

We have developed a set of tools and infrastructural components to support software engineers in the task of constructing and deploying context-aware applications that use CML models. These include a context management layer that augments a relational database with additional functionality required for storing and querying CML models, and a schema compiler toolset that supports a variety of transformations on CML models. The schema compiler tools accept a context model description in the context schema notation we discussed in Section 3.5, perform checks to verify the integrity of the model, and then produce one or more of the following outputs:

- SQL scripts to load and remove context model definitions from relational databases;
- model-specific helper classes, for Java and Python, that can be used by application developers to simplify source code concerned with context queries and updates; and
- context model interface definitions that can be compiled to stubs that can be used by applications to easily transmit or receive context information without dealing with any of the protocols used for remote communication.

The tools and infrastructure are documented further in [11] and [12].

5 Open Research Problems

Context modelling has recently become a hot topic in the field of pervasive computing, and numerous modelling approaches have appeared since we first began working on CML in 2002. However, many open research problems remain. In particular, more work is needed in relation to modelling, querying and reasoning over imperfect context information, in order to adequately address problems such as sensing errors and sensor failures. Although solutions exist in other fields for dealing with imperfect information, it is likely that no solution will provide a perfect match with the requirements of context-aware systems.

Further work is also needed to develop sophisticated context management systems, which must be radically different to the average relational database system. One important issue is traceability - that is, being able to track context information from its source, through various forms of processing (e.g., sensor fusion), to the repositories in which the information eventually resides. This kind of tracking is needed to link incorrect context information to failed or misconfigured components, thereby enabling debugging and repair. Context management systems must also address issues of scalability, performance and distribution in order to satisfy the requirements of pervasive systems, which may involve very large number of mobile sensors, applications and users. So far, only small prototypes have been developed which have not needed to address these problems.

Finally, work is needed on how to provide interoperability between multiple context-aware systems that each possess their own context models, and possibly also their own context modelling approaches. Some early work has already begun in this area using ontology standards such as DAML+OIL and OWL [13,14].

6 Concluding Remarks

In this paper, we introduced CML, an ORM-based approach for modelling the context information required by context-aware applications. To date, we have used CML to produce context models for several context-aware communication applications [15,16], a vertical handover application [11], and applications to support independent living of elderly people living in 'smart homes' [17]. With the exception of the independent living applications, all of these applications have been fully implemented. In conjunction with these efforts, we have built a suite of tools to support software engineers in building and deploying applications that use CML models, leveraging the mapping to the relational model.

Although numerous research challenges remain in relation to context modelling, we believe that CML is one of the most viable of the currently available solutions, and is well positioned to serve as a platform for investigating new modelling constructs that will begin to address these challenges. We have found ORM's fact types to provide a natural basis for extension and annotation with metadata and constraints - more so than attribute-based modelling approaches such as ER and UML, and ontology languages, such as OWL, which we have also evaluated as techniques for context modelling, as discussed in [18] and [14].

References

1. Strang, T., Linnhoff-Popien, C.: A context modeling survey. In: UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management, Nottingham (2004) 34–41
2. Halpin, T.A.: Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design. Morgan Kaufman, San Francisco (2001)
3. Pornphol, P., Chittayasothorn, S.: A temporal relational and object relational database design technique. In: SoutheastCon. (2004) 54– 59
4. Henricksen, K., Wishart, R., McFadden, T., Indulska, J.: Extending context models for privacy in pervasive computing environments. In: 2nd International Workshop on Context Modelling and Reasoning (CoMoRea), PerCom'05 Workshop Proceedings, IEEE Computer Society (2005) 20–24
5. Gregersen, H., Jensen, C.S.: Temporal entity-relationship models - a survey. IEEE Transactions on Knowledge and Data Engineering **11** (1999) 464–497
6. Wang, R., Reddy, M.P., Kon, H.: Towards quality data: An attribute-based approach. Decision Support Systems **13** (1995) 349–372

7. Jensen, C.S., et al.: The consensus glossary of temporal database concepts - February 1998 version. In: Temporal Databases: Research and Practice. Volume 1399 of Lecture Notes in Computer Science., Springer (1998) 367–405
8. Tauzovich, B.: Towards temporal extensions to the entity-relationship. In: 10th International Conference on the Entity-Relationship Approach (ER), San Mateo (1991) 163–179
9. Henricksen, K.: A Framework for Context-Aware Pervasive Computing Applications. PhD thesis, School of Information Technology and Electrical Engineering, The University of Queensland (2003)
10. Storey, V., Wang, R.: Modeling quality requirements in conceptual database design. In: 3rd Conference on Information Quality (IQ), Cambridge (1998) 64–87
11. Henricksen, K., Indulska, J., McFadden, T., Balasubramaniam, S.: Middleware for distributed context-aware systems. International Symposium on Distributed Objects and Applications (DOA) (to appear) (2005)
12. McFadden, T., Henricksen, K., Indulska, J.: Automating context-aware application development. In: UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management, Nottingham (2004) 90–95
13. Strang, T., Linnhoff-Popien, C., Frank, K.: CoOL: A Context Ontology Language to Enable Contextual Interoperability. In: 4th International Conference on Distributed Applications and Interoperable Systems (DAIS). Volume 2893 of Lecture Notes in Computer Science., Springer (2003) 236–247
14. Henricksen, K., Livingstone, S., Indulska, J.: Towards a hybrid approach to context modelling, reasoning and interoperation. In: UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management, Nottingham (2004) 54–61
15. Henricksen, K., Indulska, J.: A software engineering framework for context-aware pervasive computing. In: 2nd IEEE International Conference on Pervasive Computing and Communications (PerCom), IEEE Computer Society (2004) 77–86
16. McFadden, T., Henricksen, K., Indulska, J., Mascaro, P.: Applying a disciplined approach to the development of a context-aware communication application. In: 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom), IEEE Computer Society (2005) 300–306
17. Indulska, J., Henricksen, K., McFadden, T., Mascaro, P.: Towards a common context model for virtual community applications. In: 2nd International Conference on Smart Homes and Health Telematics (ICOST). Volume 14 of Assistive Technology Research Series., IOS Press (2004) 154–161
18. Henricksen, K., Indulska, J., Rakotonirainy, A.: Modeling context information in pervasive computing systems. In: 1st International Conference on Pervasive Computing (Pervasive). Volume 2414 of Lecture Notes in Computer Science., Springer (2002) 167–180