Automating Context-aware Application Development

Ted McFadden and Karen Henricksen
CRC for Enterprise Distributed Systems Technology (DSTC)
{mcfadden,kmh}@dstc.edu.au

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

ABSTRACT
To develop robust, evolvable, context-aware applications that are flexible enough to reconfigure and adapt in response to changes in context such as location, connectivity, resources, security, and user preferences, a rich context model, infrastructure components, and programming paradigms are required. The development and deployment of context-aware applications can be further assisted by tools to automate the transformation of the context model into appropriate forms for ready manipulation by developers and administrators. This paper describes an extensible set of tools we are developing for use in conjunction with our existing context modelling framework and infrastructure that provides substantial assistance to the development and deployment of context-aware applications.

Keywords
context modelling, context management

1. INTRODUCTION
Constructing context-aware applications is a challenging and time consuming task as the applications need to be supported by both context models and context management systems. Modeling context information, especially for complex context-aware applications, can be error prone. This is even further exacerbated by the fact that context models for context-aware applications may evolve and the consequent required changes may also introduce errors. Context management systems, on the other hand, are quite complex as they have to allow addition, removal and update of context information and need to support rich context query capabilities and notifications about context changes. Context-aware applications based on evolvable context models and on complex context management systems would benefit from automation tools which support consistency checking of models and also automate some software engineering tasks related to context management and context dissemination.

In this paper we describe automation tools developed to assist both our context management system and development of context-aware applications. The tools support the following tasks: (i) validation of the context model and automated mapping of the model to the context management system, (ii) automatic generation of context-model specific programming libraries for context-aware applications, and (iii) automatic generation of an adapter to the messaging/notification system used by context-aware applications and context management system.

*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).

The structure of the paper is as follows. In section 2 we briefly introduce our approach to context modeling and show an example context model defined for a context-aware communication application. Section 3 presents a brief description of mapping of the context model to a context management system. The remaining sections use the example introduced in section 2 to illustrate the functionality of particular automation tools. Section 4 discusses a context schema which can be derived/defined from our context model and characterises three types of our automation tools. Section 5 describes automation of the mapping of the context model to the context management system. Section 6 describes generation of supporting programming libraries whereas section 7 overviews the tool which generates an adapter to the messaging/notification system. Section 8 concludes the paper and discusses other possible automation which will further assist developers of context-aware applications.

2. CONTEXT MODELLING APPROACH
This section briefly reviews our approach to context modelling. Further information can be found in earlier papers [1, 2, 3].

We model context at two levels of detail, using fact and situation abstractions. Section 2.1 presents our fact-based context modelling approach, the Context Modelling Language (CML), while Section 2.2 describes how we represent abstract situations in terms of fact types modelled with CML. Section 3 briefly outlines a previously developed approach to mapping from CML to the relational data model, and describes how we exploited this mapping to implement a generic context management system as extensions on top of a relational database.

2.1 CML
CML primarily serves as a tool that enables application developers to explore and formally specify the context requirements of a context-aware application. It provides constructs for defining the entities about which context information is required and the types of information (or facts) that are of interest in relation to each entity. It also allows developers to identify an appropriate source for each fact type (sensors, user profiles or derivation from other context information), specify dependencies and constraints, and explore information quality issues as described in [2].

Our modelling approach builds on Object Role Modeling (ORM) [4], which provides a graphical notation designed primarily for conceptual modelling of information systems. ORM represents object types as ellipses and relations on one or more object types as fact types, drawn as sequences of role boxes (where each role is attached to the object type that participates in the role). Each object type is assigned a name as well as a reference mode, shown in parentheses, that describes how instances of the type are represented. Fact types are annotated with uniqueness constraints, represented as double-headed arrows spanning one or more role boxes; these place restrictions on the populations of the fact types in the manner of key constraints on attributes of relations in the relational data model. A variety of other constraints are also supported, but a discussion of these falls outside the scope of this paper.

CML introduces a variety of extensions to this basic notation, illustrated in an example model shown in Figure 1. The example builds on a context model that we defined for a communication application that assists users with the selection of appropriate communication channels for their interactions with other people. For this application, the most relevant types of context information include associations of users to communication channels and devices (e.g., ownership, permissions and proximity in the case of devices), current locations of users, and current and planned activities.

The CML extensions allow fact type classifications to be labelled as:

- *static* (s), *sensed* ($\wedge\!\!\wedge$), *derived* (*) or *profiled* ($\circ$) types, depending on persistence and source;
- *temporal* ([ ]) types that capture histories of context information (e.g., user activity over a period of a day or week); and
- *alternative* (a) types that are capable of describing ambiguous information (e.g., conflicting location reports gathered from a variety of location sensors).

CML also provides extensions to support special constraints on temporal and alternative fact types, annotation of fact types with appropriate metadata types, and dependencies between pairs of fact types (e.g., between a person's activity and their current location, to indicate that location changes are typically linked to activity changes).

2.2 Situation-based context modelling

In addition to modelling context at the fact level using CML, we provide a situation abstraction for describing context in high-level terms. Situations are defined using a variant of predicate logic to capture abstract classes of context in terms of the fact types of a CML model. Situations can be easily combined using logical connectives to form increasingly rich context descriptions. This feature makes them useful as programming abstractions that allow the software engineer to predicate application behaviour on simple situation expressions in a very natural way.

Situations are written as predicates on zero or more variables, then evaluated against a set of variable bindings and a context (represented as a repository of facts) to yield one of the values *true*, *false* or *possibly true*. The *possibly true* value arises when the available context information is inadequate to determine absolute truth or falsity (e.g., because of incompleteness or ambiguity). Some example situations are shown in Table 1. We presented the definitions of most of these situations and a general discussion of the situation logic in a previous paper [1], and do not repeat them here.

3. MAPPING TO A CONTEXT MANAGEMENT SYSTEM VIA THE RELATIONAL DATA MODEL

We have implemented a context management system based on our context modelling approach, which leverages a straightforward translation of CML to the relational data model. This translation builds on ORM's relational mapping procedure, Rmap [4], incorporating additional mappings for the context modelling constructs introduced by CML, to appropriate constraints and metadata, as described briefly in [5].

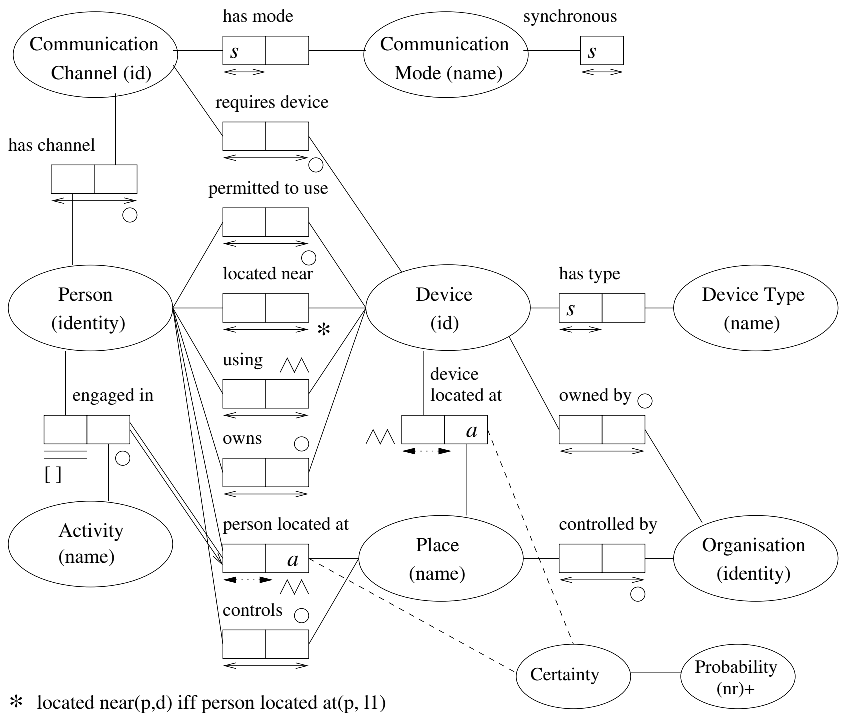
The context management system is responsible for storing the context models and their instantiations, parsing and evaluating situation expressions, processing queries and subscriptions, to receive context change notifications, formulated in terms of either facts or situations, providing transaction support for queries, and managing updates from a variety of sources in accordance with integrity constraints expressed in the CML models. The implementation is currently built on top of a relational database.

4. USING A CONTEXT SCHEMA TO AUTOMATE DEVELOPMENT

The Context Modelling Language (CML) described in Section 2.1 provides a powerful technique to capture application context information requirements in a comprehensive manner, providing a visual representation and a semi-automated procedure to map context models to context management systems based on relational databases. However, our initial experiences in developing CML-based context-aware applications suggested that the database and application coding and administration effort could be further reduced and made more error resistant by introducing additional tools to automate aspects of deploying, administering, and programming with context.

More specifically, tools are needed to address issues such as mapping context models to specific context management systems (based on SQL or other non-relational data stores), managing database and programming language namespaces, and providing programming language libraries and development tool (e.g., eclipse IDE [6]) support for manipulation of context information.

To begin addressing these issues, we developed a text-based context schema that captures CML models and the higher level situation definitions. This allows a full context model to be validated and processed by a context schema compiler. Additional tools that front-end the compiler can then transform the context models into forms useful for application development. The context schema syntax is similar to an SQL database schema [7], providing an easy transcription from CML models, and a familiar syntax to many developers. We initially identified the following automation tool outputs that



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

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

Figure 1: Modelling context for a communication application using CML.

would have directly reduced the coding effort involved in our prototype applications:

- SQL scripts to load and remove context model definitions from the context management system database.
- Context-model specific Java (and other programming language) classes to allow context type safety checking at compile time, to feed existing development tool features such as method selection code completion, and to eliminate manual coding of repetitious, boiler-plate, context manipulation code.
- Support for sending context information as notifications over a content-based publish-subscribe notification router (such as Elvin [8]) for use with the context management system and any other loosely coupled context producers and consumers in the system.

Figure 2 shows selected portions of the context schema for the context model shown in Section 2. The `CREATE SCHEMA` statement defines the context schema namespace, which allows for deploying multiple context models in a single context management domain. Context schemas may reference elements from other schemas by fully scoping the reference. The schema naming scope also partially defines the output scope of other programming artifacts such as Java package names. The `DOMAIN` statements that follow define the type names used in the context model. The domain statements that map to base SQL data types are not shown due to space limitations. Next are the `FACT` statements which capture fact names, their classifications (static, profiled, derived, sensed, alternative, temporal) , dependencies, and details of context fact information, such as the alternative role (`ALTROLE`) indicator for the `location` attribute of the `CurrentLocation` fact type. Finally, the context schema supports the definition of situations through the `SITUATION` statement. The first portion of the statement is similar to an SQL `CREATE TABLE` statement. The remainder of the situation definition syntax is defined in [1].

The context schema syntax described is sufficient to fully capture our context model definitions. Figure 3 illustrates the tool architecture used to process context schema definitions. The first tool we developed was the context schema compiler to validate and process context models into a common intermediary form. From this, task specific front end tools generate specific outputs. The tools developed to address our initial requirements are also shown in the figure. They are described in more detail in the following sections.

5. MAPPING TO CONTEXT MANAGEMENT SYSTEMS

In our initial work, to map context models into corresponding database relations as required by our SQL based context management system, we relied on simple (manually produced) Java programs. These programs generated and executed appropriate SQL statements via the JDBC API.

Although fully functional, these programs required careful

Table 1: Example situations for the context-aware communication application.

Situation	Description
Occupied(person)	True whenever *person* is currently involved in an activity such as a meeting or phone call, as determined by the `EngagedIn` fact type from our example model.
CanUseChannel(person, channel)	True whenever *person* has appropriate access to, and permissions to use, the devices associated with *channel*, as determined by the `RequiresDevice`, `LocatedNear` and `PermittedToUse` fact types.
Engaged(device)	True whenever *device* is a telephone, according to the `HasType` fact type, and is currently being used by at least one person, according to the `Using` fact type.
WorkingHours()	Indicates whether the current time falls within normal working hours.

```
CREATE CONTEXT SCHEMA DSTC.PACE.COMM
...
CREATE DOMAIN PersonID AS Identity...
CREATE DOMAIN ChannelID AS Identity...

CREATE PROFILED FACT TYPE PersonHasChannel(
 KEY(
 person PersonID,
 channel ChannelID
 )
)
...
CREATE ALT SENSED FACT TYPE
PersonLocatedAtPlace
 QUALITY(Certainty) (
 person PersonID KEY,
 place PlaceName ALTROLE
)
...
CREATE FACT TYPE PersonLocatedNearDevice
 DERIVED(person PersonID KEY,
        device DeviceID KEY)
  IFF PersonLocatedAt(person, location1)
     AND DeviceLocatedAt(device, location2)
     AND location1 = location2
...
CREATE PROFILED TEMPORAL FACT TYPE
PersonEngagedInActivity
 DEPENDS(PersonLocatedAt) (
    person PersonID KEY,
    activity ActivityName
)
...
CREATE SITUATION Occupied(person PersonID):
 exists activity .
  PersonEngagedInActivity[person,activity] .
                activity = ``meeting'' or
                activity = ``on.phone''
```

Figure 2: Selected Portions of Context Schema for Example Context Model.

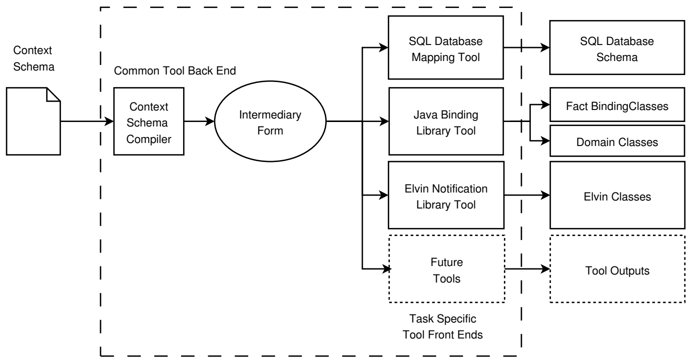


Figure 3: Tool Architecture.

coding, and had to be manually updated when a context model changed, which is not uncommon during research and development. Certain errors in naming fact type tables or other fact specifications could go undetected until client applications encountered errors manipulating context information. For these reasons, we developed an SQL database tool to automate the mapping of the context model to our existing context manager's underlying SQL database.

The abbreviated database tool output for the communication context model is shown in Figure 4. As shown first in the `CREATE SCHEMA` statement, the multi-level namespace of the original context schema has been collapsed to reflect the capabilities of the target database. Context domains and fact types are mapped to SQL domains and tables respectively. The object types involved in fact type roles map to table columns. Fact type quality annotations are mapped as additional table columns. Temporal fact types are mapped to tables wtih two additional columns to represent start and end times. Key constraints for database tables representing alternative and temporal fact types are set to support the fact type semantics in the relational domain. Derived fact types are defined as SQL views with the appropriate view `SELECT` statement. Context schema situation definitions are loaded into tables that are part of the context management system itself, as are fact metadata (not shown.)

The SQL database tool has brought direct benefit by removing manual administrative database coding, allowing rapid deployment of context models to the context management system, and ensuring context model updates are correctly applied.

We expect to be able to apply similar context management mapping tools as we explore alternatives to SQL based context managers for pervasive environments and distributed context management.

6. CONTEXT MODEL SPECIFIC PROGRAMMING LIBRARIES

The functionality of our current, Java-based, context management system is described in Section 3. The context management API provided allows applications full access (with appropriate authorisation) to context information, but as it is a generic context management interface, context model specific operations or compile time validation of request parameters are not available. As we found in practice, lack of these features can sometimes lead to application coding er-

```
CREATE SCHEMA DSTC_PACE_COMM;

CREATE DOMAIN DSTC_PACE_COMM.PersonID
        AS VARCHAR(255)...

CREATE TABLE DSTC_PACE_COMM.PersonHasChannel(
    person DSTC_PACE_COMM.PersonID,
    channel DSTC_PACE_COMM.ChannelID,
    PRIMARY KEY (person, channel)
);
CREATE TABLE
DSTC_PACE_COMM.PersonLocatedAtPlace(
    person DSTC_PACE_COMM.PersonID,
    place DSTC_PACE_COMM.PlaceName,
    qCertainty DSTC_PACE_COMM.Certainty,
    PRIMARY KEY (person, place)
);
CREATE VIEW
DSTC_PACE_COMM.PersonLocatedNearDevice (
        person, device)
    AS SELECT DISTINCT person , device FROM ....

CREATE TABLE
DSTC_PACE_COMM.PersonEngagedInActivity(
    person DSTC_PACE_COMM.PersonID,
    activity DSTC_PACE_COMM.ActivityName,
    fStartTime TIMESTAMP,
    fEndTime TIMESTAMP,
    PRIMARY KEY (person, fStartTime)
);
```

Figure 4: Context Schema to Relational Database Mapping Example.

rors that result in subtle alterations in system behavior that may not be readily identifiable without further, detailed review. After experiencing this in the first prototype systems, it became desirable to write context model specific code libraries to provide programming convenience and compile-time checking. This brought additional benefits to developers using Java IDEs, which could now offer context model specific assistive information.

The code in the first manually constructed context model specific library provided a context manipulation class for each fact type and situation as well as a class to represent each of the context model domain types (e.g., PersonName, CommunicationMode). These classes were conceptually simple, numerous, and tedious to write, making them prime candidates for automatic generation. (The communications context model presented in Section 2.1 requires over 30 such classes.)

A context model library tool was written to generate the necessary Java helper classes. As a result of using this tool, helper classes are now of a consistent form, offer complete coverage of specific context models, and can be refreshed immediately to reflect additions to context models or changes to the desired class implementations.

7. CONTEXT NOTIFICATIONS

The context-aware infrastructure components and applications make use of the previously mentioned content-based notification routing system, Elvin, which offers loosely coupled, publish-subscribe communication semantics. This style of communication is attractive in pervasive environments as it minimises dependencies on well known endpoints and allows more flexible modes of communication besides the one-to-one of RPC mechanisms such as Java RMI.

To use Elvin with the initial prototype applications, interface classes were written to send and receive context fact events over Elvin. Additionally, as the existing context manager API is not exposed via Elvin messaging, Elvin adapter applications were written to receive Elvin context events and relay them to the context manager through its Java RMI interface.

As in the case of the context model specific programming libraries, the Elvin interface classes and gateway code were boiler-plate in nature, and an Elvin notification library tool was written to generate them automatically.

This tool is shown in more detail in Figure 5. The tool is fed with the output of the context schema compiler and as a first step generates a java interface that defines methods for all of the context fact events represented by the context schema. For example, a method signature in this interface for the described communications context model is:

```
void personUsingDevice(PersonID aPerson,
                       DeviceID aDevice);
```

The generated interface method parameter types (e.g., PersonID) reference the domain classes that were generated by the Java context model library tool (shown as an input in Figure 5.) The `void` return type is indicative that this method represents an event notification and not an RPC call.

The generated context fact interface is then processed by an additional tool we have developed, the Elvin interface compiler. This tool takes a Java interface and generates send and receive stubs that map interface methods to Elvin notifications. This eliminates the need for applications to manually construct and unpack Elvin notification messages. Java and Python are the currently supported target languages. After this step, Elvin stubs are available for the context fact interface.

Finally the Elvin notification library tool uses the just created Elvin receiver stub to generate a context model specific Elvin adapter for the context management system. This class provides a gateway to relay Elvin context fact messages to the context manager. It uses the fact binding and domain classes generated by the Java context model library tool. It may be run as part of the context manager process or as a stand alone gateway.

This tool allows any context model to be instantly accessible to Elvin clients written in a variety of languages running on a number of operating system platforms. We expect to develop further Elvin generation tools as we extend the capabilities of the context management system.

The Elvin generation tool model is extensible to a number of other distributed communication infrastructures, RPC or notification based, such as CORBA or web services (WSDL). This will give us further opportunities to extend the context mangement system operating domain.

8. CONCLUSIONS

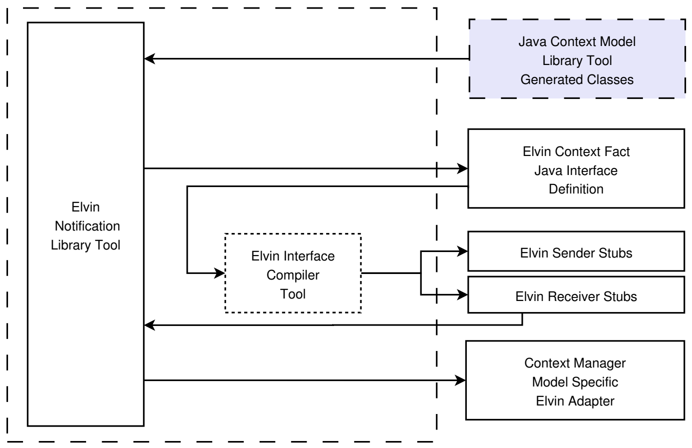

Java Context Model Library Tool Generated Classes
Elvin Context Fact Java Interface Definition
Elvin Notification Library Tool
Elvin Interface Compiler Tool
Elvin Sender Stubs
Elvin Receiver Stubs
Context Manager Model Specific Elvin Adapter

Figure 5: Elvin Notification Library Tool Detail.

As our early experiences have shown, context-aware application development requires significant and careful coding efforts to ensure that context models are faithfully represented in the target context management system and that applications perform valid manipulations of context information. Any changes to a context model requires detailed re-examination of context-related code.

To address the limitations of manually constructing context manipulation code, we adopted a model driven development approach. A CML-based context model is expressed as a text-based context schema. The context schema is validated and processed into an intermediary representation by a context schema compiler. From this form, specialised front-end tools generate specialized outputs. In this paper we have described tools to generate SQL database scripts for our context management system, context model specific programming libraries, and adapters to the Elvin notification router. Each of the tools has contributed to reducing the effort required to develop and maintain context-aware applications.

The paper also discussed extending the tools to add new features to context libraries, adapters to other communication transports, and support for alternative context management systems. We are currently investigating extending the features of the context model specific library tool, and exploring ontology-based approaches to context modelling by creating tools to represent CML-based models in ontology languages such as OWL DL [9] and SWRL [10].

REFERENCES

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

[2] Henricksen, K., Indulska, J.: Modelling and using imperfect context information. In: Workshop on Context Modeling and Reasoning (CoMoRea), 2nd IEEE Conference on Pervasive Computing and Communications (PerCom), Orlando (2004) 33–37

[3] 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

[4] Halpin, T.A.: Conceptual Schema and Relational Database Design. 2nd edn. Prentice Hall Australia, Sydney (1995)

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

[6] Foundation, E.: Eclipse: Open extensible integrated development environment (2004) http://www.eclipse.org.

[7] ANSI/ISO/IEC: Structured Query Language (SQL) Standards. (2003) ISO/IEC Standards 9075-1 – 9075-5.

[8] Segal, B., Arnold, D., Boot, J., Henderson, M., Phelps, T.: Content based routing with elvin4. In: Proceedings of the AUUG2K Conference. (2000)

[9] McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language - Overview. W3C Recommendation (2004)

[10] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A semantic web rule language combining OWL and RuleML, Version 0.5 (2003) http://www.daml.org/2003/11/swrl.