XCML: A runtime representation for the Context Modelling Language

Ricky Robinson and Karen Henricksen
National ICT Australia
Queensland Research Laboratory
{ricky.robinson,karen.henricksen}@nicta.com.au

Jadwiga Indulska
School of Information Technology and Electrical Engineering
University of Queensland
and
National ICT Australia
jaga@itee.uq.edu.au

Abstract

The Context Modelling Language (CML), derived from Object Role Modeling (ORM), is a powerful approach for capturing the pertinent object types and relationships between those types in context-aware applications. Its support for data quality metrics, context histories and fact type classifications make it an ideal design tool for context-aware systems. However, CML currently lacks a suitable representation for exchanging context models and instances in distributed systems. A runtime representation can be used by context-aware applications and supporting infrastructure to exchange context information and models between distributed components, and it can be used as the storage representation when relational database facilities are not present. This paper shows the benefits of using CML for modelling context as compared to commonly used RDF/OWL-based context models, shows that translations of CML to RDF or OWL are lossy, discusses existing techniques for serialising ORM models, and presents an alternative XML-based representation for CML called XCML.

1. Introduction

Software engineering processes are, in general, constituted by a number of distinct phases. These include requirements gathering and analysis, design and implementation. These are ordinarily followed by some kind of verification or testing step, and maintainence. Depending on the particular process, these phases may be repeated. Various tools and methodologies are used during and between these phases to formalise and, in many cases, simplify the task of converting a set of initial requirements into a concrete, operational system. Conceptual models are often used to capture the design of a system. Good modelling notations simplify the task of converting domain knowledge into a system "blueprint". Tools also exist to fully or partially automate the process of converting conceptual models into concrete implementations. We argue that the engineering of context-aware systems should be no different to traditional software engineering in these respects. The Context Modelling Language (CML) has proven to be an excellent graphical notation for capturing context information requirements at design time [8, 9]. The process of mapping a CML model to a relational database has also been described previously. However, in distributed systems, there is also a need to be able to share context information and context models between loosely coupled entities.

This paper shows why CML is a better choice for modelling context information, specifically at the design stage, than RDF and OWL based approaches. It also introduces an XML serialisation of CML, called XCML, for sharing context information among distributed entities, and it justifies the introduction of this new serialisation by showing that CML and context facts that conform to CML cannot be adequately translated to XML Schema-based or OWL-based representations.

The remainder of this paper is structured as follows. Section 2 provides a critique of two modelling techniques that are commonly used in context-aware systems: RDF and OWL. It shows that the modelling constructs of CML are considerably more natural than those of RDF and OWL, validating our decision to model context information with

CML. Section 3 provides a brief overview of two previously proposed XML representations of ORM [6, 7] (the modelling notation from which CML is derived), highlighting the differences between these representations and the approach that we propose in this paper. Section 4 presents the design of our solution and a small modelling example. Section 5 gives an overview of related work in this area. Finally, Section 6 summarises the contributions of this paper and discusses future work.

2. Critique of RDF and OWL

RDF and OWL are gaining popularity as means to model context information [3, 4, 14]. We argue that RDF and, by extension, OWL suffer important limitations that restrict their effective use as context modelling formalisms.

First, RDF supports only binary relations. This has the implication that any relation in the domain being modelled with an arity not equal to two must be translated into a set of binary relations. The immediate effect of this is that RDF/OWL representations of context are a step further removed from the universe of discourse. Higher arity relations are necessary to simplify the process of *correctly* capturing the domain requirements, and ensuring that the captured model closely resembles the domain being modelled. Further problems follow from the necessity of converting n-ary relations to their binary equivalent. In translating an n-ary relationship, the modeller must make a decision as to how to decompose (or, in the case of unary relations, to synthesize) the relation into a set of binary relationships. For unary relations this is simple: the binary equivalent relates a subject to a boolean value. But for relationships of arity greater than two, there are several choices [12]. The problem is that RDF and OWL are inconsistent in the way they model n-ary relationships, because the translation to a set of binary relationships depends upon the particular use case. Furthermore, relationships of arity greater than two always have a different representation to binary relationships because they all involve objectifying the relationship. Finally, if the original n-ary relationship, as expressed in a conceptual modelling language capable of representing n-ary relations directly, is to be reconstructed from the RDF representation, then the RDF representation must include additional information indicating that one or more objects in the RDF model are, in fact, objectified relations. This additional information adds further complexity to the RDF model, but it allows the original and intended semantics to be restored.

In OWL, there is a tendency to define classes as subclasses of anonymous classes that assert restrictions on certain properties. Figure 1 shows an example of this: *Vehicle* is a subclass of the (anonymous) class of things that have one or more wheels. The use of an anonymous class in this example is an artefact of the manner in which such a definition is usually serialised:

```
<owl:Class rdf:about='#Vehicle'>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource='#numberOfWheels' />
      <owl:minCardinality
        rdf:datatype='&xsd;nonNegativeInteger'>
        1
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The anonymous restriction class is specific to the *Vehicle* class and cannot be referenced by other classes. This means that other classes that have one or more wheels must define their own anonymous restriction class. This heavy use of the subclass property and of restriction classes means that OWL models quickly become unwieldly. OWL models are rarely shown using the graphical RDF representation as in Figure 1. Rather, the XML serialisation is more commonly used, indicating that there is little difference in the level of clarity between the graphical and serialised representations. We contend that this may retard the uptake of OWL, because designers prefer to use easily understood graphical models that more directly and closely represent the domain being modelled. This problem extends to the modelling of context information.

ORM and ORM-based representations, such as CML, do not suffer from these difficulties, which makes them a better choice for modelling context information. Furthermore, some constraints that can be expressed in CML are difficult to translate to OWL, unless a rule language such as the Semantic Web Rule Language (SWRL) [10] is used on top of OWL, which adds a further layer of complexity. For example, the external uniqueness contraints of ORM and CML cannot be expressed succinctly in OWL. That is to say, it is difficult to express the case where individuals of a class C can be uniquely identified by the combination of two (or more) of its properties. This may be achievable in OWL by objectifying the combination of properties into a class P, and then defining a new property from C to P and applying the appropriate cardinality constraint on this new property. However, this manipulation far removes the model from the domain problem, and therefore does not lend itself to simple capture of the domain requirements.

3. Mappings of ORM to XML

The XML representation of CML discussed in the next section is the first attempt to serialise CML models and CML conformant context facts. However, there are already several solutions for representing *ORM* models in XML. In this section, we show why these XML serialisations for ORM are not applicable to CML.

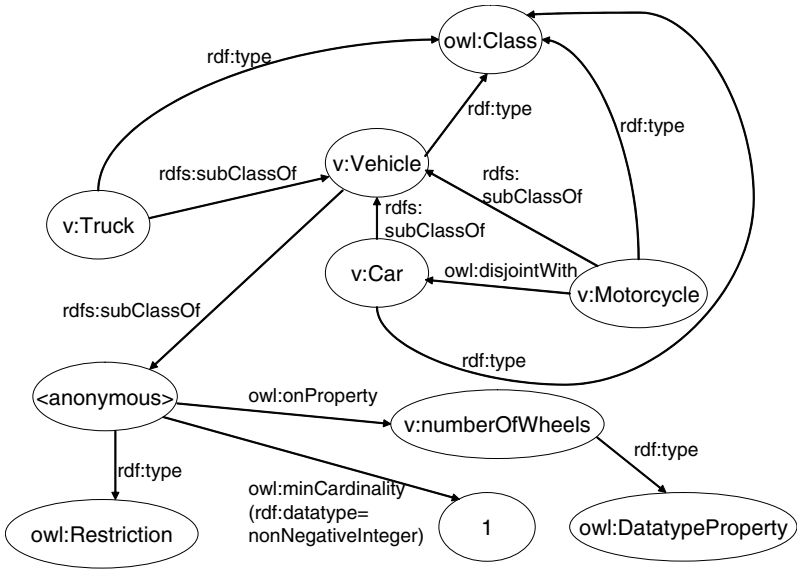

Figure 1. Graphical representation of a simple OWL ontology for vehicles.

The earliest solution (that we are aware of) for transforming ORM models to XML was proposed by Bird et al. [2]. They showed that it is possible to automatically map ORM models to XML Schema, effectively creating a unique XML language from each model. In this approach, instances (facts) can be exchanged in heterogeneous systems using the unique XML language, and ORM models can be exchanged as XML Schema documents. However, mappings from ORM models to XML Schemas are lossy (e.g., some constraints cannot be captured) so it is not possible to fully reconstruct an ORM model from an XML Schema.

Bird et al.'s mapping algorithm focuses on producing a human-readable, hierarchical XML document in which redundancy of information is minimised. As ORM models are not inherently hierarchical, Bird et al. evaluate the relative importance of object types participating in fact types and then structure the hierarchy accordingly.

Several characteristics of this mapping approach make it unsuitable for our purposes. As mentioned above, it provides an imperfect solution for exchanging ORM models. Additionally, the nested structure of the instance documents hinders the exchange and reassembly of information in piecewise fashion, as is often necessary in a context-aware system. Specifically, context-aware systems require the ability to exchange any number of facts at a time, and later reassemble these into larger instance documents without the difficulty of reconstructing hierarchies. A further problem, again resulting from the hierarchical structure, is the difficulty of merging models (and any legacy instance documents).

In the XML representation that we propose in the following section, which adopts a flat document structure, these issues become relatively straightforward. In addition, models and instances are all described using a single construct, which makes it possible to (partially) validate and interpret them without any prior knowledge. This is not possible with Bird et al.'s approach, as they create an XML language for each ORM model.

Another serialisation of ORM was proposed by Demey et al. [5]. This solution, called ORM-ML, is an XML language for capturing ORM models in a lossless way, so that models can be completely reconstructed from the XML description (barring stylistic aspects, which can be captured by separate stylesheets). ORM-ML provides an excellent solution for exporting models from one ORM CASE tool to another, for example. However, as it does not address instances, it is not suitable for our purposes. Although we could potentially adopt ORM-ML for exchanging models, and create a separate solution for exchanging instances, we have instead chosen to create a single, uniform representation for both, as this approach is conceptually cleaner and requires only a single parser/validator. In this respect, our XML representation of CML is similar to RDF, which also utilises a single representation for schemas and instances.

4. An XML Mapping of CML-based Context Models

4.1. Design Rationale

Our XML representation of CML, XCML, was designed with the following goals in mind. First, it should be easy to read. Although designers and developers should primar-

ily use the graphical CML notation, it is inevitable that they will occasionally need to deal with the textual representation directly. Next, the serialisation must consist of a small set of syntactical constructs, thereby reducing the learning curve for developers and minimising the difficulty of writing a parser/validator for the language. For similar reasons, the *schema* language should be the same as the *instance* language. The serialisation should coincide as closely as possible with the graphical model. That is, the mapping ought to be direct, with no need to group fact types as is required by the ORM to XML Schema mapping (see Section 3). This property simplifies reconstruction of the graphical model from the serialised version. The representation must enable new fact types to be added after the original model has been developed. This must be achievable without requiring the graphical model to be retranslated. It must aid the representation and communication of individual facts. Finally, it should be capable of representing constraints and properties, such as entity type and fact type equivalence, that are not currently part of the ORM or CML specifications, but which become important in distributed, context-aware applications in which many different context models may coexist.

These goals led to the development of a serialisation that is somewhat different from XML Schema and RDF.

In XCML, everything is represented as a *fact*. CML models are described using facts which are instances of one of a set of pre-defined fact types. These pre-defined fact types are the result of serialising the CML meta-model in XCML. Thus, XCML is self-defined, similarly to RDF. XCML therefore has the property that CML models and populated instances are represented using the same constructs. The only difference between model documents and instance documents is that models are enclosed within the *model* tag and fact instances are enclosed within the *factBase* tag. The result of parsing an XCML model document is an ORM/CML model rather than a document object model (DOM) as in the case of standard XML approaches. Parsing an XCML fact base yields a set of facts that can be validated against the relevant model. XCML facts have the following general structure:

```
<fact-type-name>
  <role-name1>role value 1</role-name1>
  <role-name2>role value 2</role-name2>
  ...
</fact-type-name>
```

Since XCML represents everything as facts, and because facts cannot be nested, the mapping procedure is much simpler than that required by the ORM to XML Schema approach, described in the previous section. The result is a rendering that corresponds closely to the original ORM/CML diagram. It also means that additional fact types can be inserted into the serialised model easily, without disturbing existing fact types. In other words, the model can be extended without the need to re-map the graphical model to its serialised form.

Another advantage of XCML is that it enables individual facts to be represented and transmitted easily. In RDF, properties are encapsulated within classes and objects; therefore facts are a combination of the encapsulating class, the name of the property and the property value. In the XML Schema mapping, each model generates a hierarchical structure so that individual facts may be encoded within several levels of nesting. In XCML, facts are represented more naturally, whereby the fact construct encapsulates the associated object references or values, which are in turn declared via other facts. XCML enables facts to be parsed and understood at a basic level independently of the schema (model). This is by virtue of the fact that there is a single construct in XCML: the fact. In the ORM to XML Schema mapping, each ORM model generates its own language with a unique structure, which means that facts conforming to a particular model cannot be recognised as facts by a remote entity unless that entity can refer to the schema document. This is an issue of syntax. In XCML, the model document is required only to gain knowledge about what other roles might be played by the objects in the current fact and to check constraints.

These features make XCML relatively easy to read. In fact, reading an XCML document is rather like reading a natural language document in which each sentence conveys an elementary fact. The overall meaning of an XCML document is built up by reading individual facts in the same way that the overall meaning of a natural language document is established by reading individual sentences. This format is similar to the manner in which ORM designers are encouraged to verbalise elementary facts from the universe of discourse as the first step in creating an ORM model. It is possible, therefore, that an ORM/CML model could be partially derived from a set of elementary facts expressed using XCML (which requires little effort over and above expressing the facts in natural language).

XCML introduces some elements that are not defined in CML or ORM. The first element is the assertion of equivalence between two entity types or fact types. This element exists so that links can be formed between multiple models that have been developed by different people or organisations. This expands the reach of any process that uses CML facts as input (such as a reasoner, agent or context management system), because it has the potential to increase the number of facts under consideration. XCML also introduces an implicit type hierarchy, where unless a model explicitly states otherwise, all types inherit from the predefined *Object* type. The benefit of this is that the existence of all objects, regardless of their type, is asserted using the same fact type, and this fact type declares an object to have a name (*id*) and a type. The newly declared object can then be

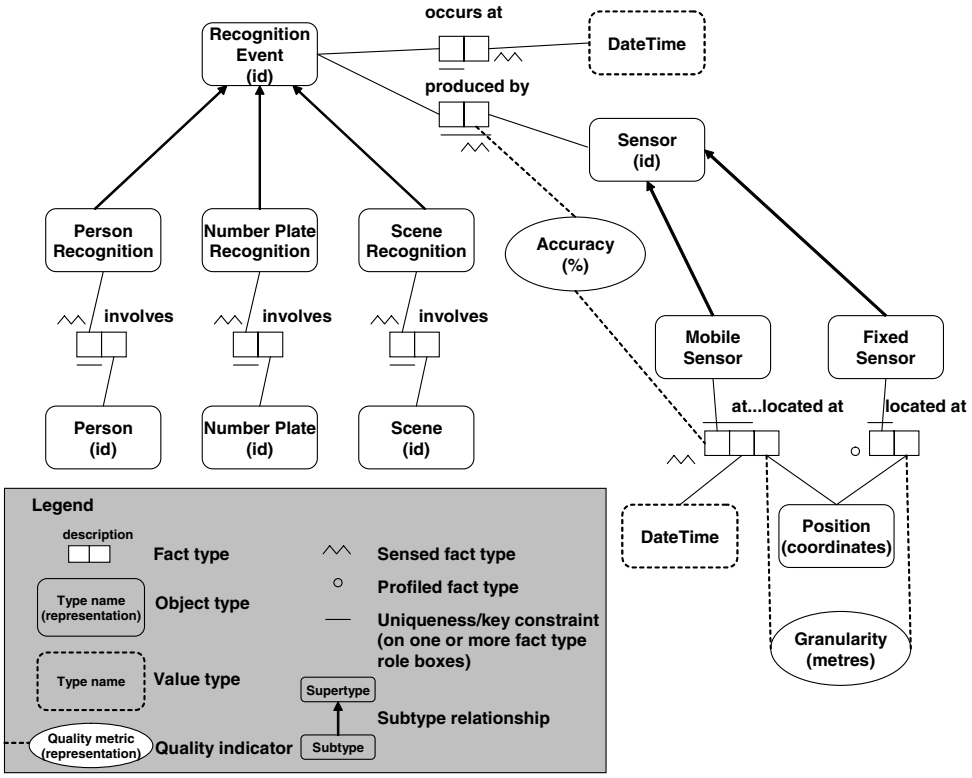

Figure 2. A CML model for a surveillance application.

referenced from any XCML document via its name. XCML names conform to the XML Namespace syntax for qualified names, though unlike existing XML-based languages, names can appear in character data sections in addition to being used as element names. Thus, within XCML documents, objects are referred to via a URI. To introduce a new type (or fact type), an object is declared whose type is *Type* (or *FactType*). The section below shows an example of this.

The ability to represent hierarchies of types or concepts and to assert relationships between the types, coupled with a simple textual representation geared towards exchanging facts in distributed environments, means XCML is an excellent candidate for representing ontological information. ORM/CML and XCML can therefore be viewed as alternatives to RDF/OWL and the RDF/XML serialisation.

4.2. XCML Example

Figure 2 shows a CML diagram that contains entity types and fact types relevant to a surveillance application. Each fact type is marked as being either profiled or sensed, according to the expected sources of facts for each fact type. In addition, three of the fact types have attached quality metrics which indicate the accuracy and granularity of fact instances. Figure 3 shows a partial serialisation of the CML model depicted in Figure 2. The full serialisation can be found on the web[1].

The *Declaration* fact type can be read as *Object has Type*. All objects are defined through this fact type. Objects are then referred to via their name (*id* role). The *Declaration* fact type appears in XCML model documents, where it is used to define the types, fact types, constraints, etc., that constitute the model, and in fact bases, where it is used to assert the existence of objects whose type has been previously defined in a model document. The *Role* fact type can be read as *Type plays Role in Fact Type*. This fact type associates types with the roles they play in various fact types. *RefScheme* is similar to the *Role* fact type, except that it asserts that the role being defined is part of the reference scheme for the type playing that role. Figure 4 shows some examples of fact instances rendered in XCML. Figure 3 gives the general flavour of XCML, but it omits many of the declarations, the definition of the quality metrics and the uniqueness constraints (for these, see the full model document on the web).

[1] http://cml.randomresearch.net/xcml/surveillance

```
<xcml:model
  xmlns:xcml='http://nicta.com.au/xcml'
  xmlns:s='http://example.com/surveillance'>
  <!-- Define the entity types ------------------>
  <xcml:Declaration>
    <id>s:Sensor</id>
    <type>xcml:EntityType</type>
  </xcml:Declaration>
  <xcml:SubtypeOf>
    <subtype>s:MobileSensor</subtype>
    <supertype>s:Sensor</supertype>
  </xcml:SubtypeOf>
  <!-- Define the fact types -------------------->
  <xcml:Declaration>
    <id>s:MobileSensorAtTimeLocatedAtPosition</id>
    <type>xcml:FactType</type>
  </xcml:Declaration>
  <xcml:Role>
    <type>s:MobileSensor</type>
    <role>sensor</role>
    <factType>
      s:MobileSensorAtTimeLocatedAtPosition
    </factType>
  </xcml:Role>
  <xcml:Role>
    <type>xcml:DateTime</type>
    <role>time</role>
    <factType>
      s:MobileSensorAtTimeLocatedAtPosition
    </factType>
  </xcml:Role>
  <xcml:Role>
    <type>s:Position</type>
    <role>position</role>
    <factType>
      s:MobileSensorAtTimeLocatedAtPosition
    </factType>
  </xcml:Role>
  <xcml:Role>
    <type>s:Position</type>
    <role>position</role>
    <factType>
      s:PositionIdentifiedByCoordinates
    </factType>
  </xcml:Role>
  <xcml:Role>
    <type>xcml:integer</type>
    <role>id</role>
    <factType>s:SensorIdentifiedById</factType>
  </xcml:Role>
  <xcml:RefScheme>
    <type>s:Sensor</type>
    <role>sensor</role>
    <factType>s:SensorIdentifiedById</factType>
  </xcml:RefScheme>
</xcml:model>
```

Figure 3. A partial serialisation of the model in Figure 2.

```
<xcml:Declaration>
  <id>example:MSensor1</id>
  <type>s:MobileSensor</type>
</xcml:Declaration>
<s:SensorIdentifiedById>
  <sensor>example:MSensor1</sensor>
  <id>123</id>
</s:SensorIdentifiedById>
<s:MobileSensorAtTimeLocatedAtPosition>
  <sensor>example:MSensor1</sensor>
  <time>Mon Jul 10 10:29:58 AEST 2006</time>
  <position>example:Pos1</position>
</s:MobileSensorAtTimeLocatedAtPosition>
```

Figure 4. A partial serialisation of an instance conforming to the model of Figure 2.

4.3. Querying and Reasoning

XCML is intended primarily as an XML exchange format; its design does not limit the querying and reasoning support for CML. There are a variety of query languages (as well as other tool support) for ORM, which can be leveraged for querying of instances of CML models. In addition, Henricksen et al. [8] have developed a situation logic which can be used in conjunction with CML to support high-level context querying and reasoning. This incorporates support for uncertainty and efficient forms of universal and existential quantification. However, one form of reasoning that is not currently supported by the situation logic is reasoning over multiple context models, which can require mapping of concepts from one model to another. As XCML introduces the ability to assert relationships between concepts in separate context models, it can be leveraged to add this additional form of reasoning in XCML-based context management systems.

5. Related Work

A significant body of previous work has been done on modelling context using a variety of ontology languages, including RDF, OWL and DAML+OIL. One early approach for Web-based applications to exchange context information was CC/PP [11], which was standardised as a W3C recommendation during 1998-2004. CC/PP focused on the simple kinds of context information and user preferences needed to support Web content adaptation for mobile devices, and did not address issues such as capturing data quality or supporting reasoning.

One of the earliest modelling approaches to address interoperability and reasoning was the Context Ontology Language (CoOL), developed by Strang et al. [13], which provided a simple "Aspect-Scale-Context" conceptual model with mappings into OWL, DAML+OIL and F-Logic. CoOL supports mapping of information between context models using inter-ontology relationships, and as well as reasoning to "complete" an ontology by computing implicit relationships, such as subclass relationships. Other more recent work that supports reasoning and interoperability includes Wang et al.'s OWL-based context modelling approach [14] and Chen et al.'s set of ontologies (SOUPA) [4] which define a broad, reusable set of context types. These solutions all facilitate the exchange of context information in heterogeneous distributed systems, but do not have corresponding high-level graphical representations that are suitable for design purposes (whereas CML is designed with the high-level modelling constructs as a primary focus).

Finally, Bauer et al. [1] have defined the Augmented World Modeling Language (AWML) and the accompanying Augmented World Query Language (AWQL). These lan-

guages are defined using XML Schema. AWML enables objects within the physical world to be described using coordinates, names, addresses relationships and so on. AWQL is the method by which AWML repositories can be queried. The response to an AWQL query is a set of matching objects described in AWML. Because AWML is defined using XML Schema, which does not support concepts such as multiple inheritance, some object properties must be modelled externally with another schema language. It is unclear exactly what kinds of constraints (e.g., many-to-many, ring, and subset) can be modelled with AWML (or the external schema language). As with CoOL and SOUPA, AWML was designed primarily to facilitate exchange of context information and, to our knowledge, there is no high-level representation specifically for design purposes.

6. Conclusions and Future Work

XCML provides several features that make it well suited for representing context information in distributed environments. Foremost among these features is the trivial mapping to and from the graphical CML notation. This is coupled with a range of properties that allow models represented in XCML to be easily extended and merged without having to re-serialise the graphical model. Since CML is a superset of ORM, XCML can be used to serialise ORM diagrams as well. We have developed an XCML parser/-validator in J2SE 1.5.0 with support for the more common ORM/CML constraints. We are now developing a platform for distributing context models and information, in XCML form, throughout a pervasive computing environment.

We are working to define the CML meta-model (i.e., the base XCML document) in terms of XML Schema, so that light-weight processes can validate an XCML model document with standard XML tools. This is useful where it may not be feasible to build an in-memory ORM/CML model, but where it is still necessary to check whether an XCML model is valid. In addition, we are investigating methods of automatically generating XML Schema documents from XCML model documents (using XSLT), so that XCML instance documents can also be partially validated with standard XML tools.

7. Acknowledgements

National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology, and the Arts; the Australian Research Council through Backing Australia's Ability and the ICT Research Centre of Excellence programs; and the Queensland Government.

References

[1] M. Bauer, C. Becker, J. Hähner, and G. Schiele. ContextCube - providing context information ubiquitously. In *23rd International Conference on Distributed Computing Systems Workshops*, pages 308–313. IEEE Computer Society, May 2003.

[2] L. Bird, A. Goodchild, and T. Halpin. Object Role Modelling and XML-Schema. In *19th International Conference on Conceptual Modeling (ER)*, volume 1920 of *Lecture Notes in Computer Science*, pages 309–322. Springer, 2000.

[3] S. Buchholz, T. Hamann, and G. Hubsch. Comprehensive Structured Context Profiles (CSCP): Design and experiences. In *1st Workshop on Context Modeling and Reasoning (CoMoRea), PerCom'04 Workshop Proceedings*, pages 43–47. IEEE Computer Society, March 2004.

[4] H. Chen, T. Finin, and A. Joshi. *The SOUPA Ontology for Pervasive Computing*, pages 233–258. Ontologies for Agents: Theory and Experiences. Birkhäuser Basel, 2005.

[5] J. Demey, M. Jarrar, and R. Meersman. A markup language for ORM business rules. In *International Workshop on Rule Markup Languages for Business Rules on the Semantic Web (RuleML)*, Sardinia, June 2002.

[6] T. A. Halpin. *Conceptual Schema and Relational Database Design*. Prentice Hall Australia, Sydney, 2nd edition, 1995.

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

[8] K. Henricksen and J. Indulska. Developing context-aware pervasive computing applications: Models and approach. *Journal of Pervasive and Mobile Computing*, 2(1):37–64, Feb. 2006.

[9] K. Henricksen, J. Indulska, and T. McFadden. Modeling context information with ORM. In *OTM Federated Conferences Workshop on Object-Role Modeling (ORM)*, volume 3762 of *Lecture Notes in Computer Science*, pages 626–635. Springer, November 2005.

[10] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, 2004.

[11] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M. H. Butler, and L. Tran. Composite capability/preference profiles (CC/PP): Structure and vocabularies 1.0. W3C Recommendation, 15 January 2004.

[12] N. Noy and A. Rector. Defining n-ary relations on the semantic web. W3C Note, July 2006.

[13] T. Strang, C. Linnhoff-Popien, and K. Frank. 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*, pages 236–247. Springer, 2003.

[14] Z. Wang, D. Zhang, T. Gu, J. Dong, and H. K. Pung. Ontology based context modeling and reasoning using OWL. In *Workshop on Context Modeling and Reasoning (CoMoRea), PerCom'04 Workshop Proceedings*, pages 18–22, Orlando, March 2004. IEEE Computer Society.