A Conceptual Markup Language that Supports Interoperability between Business Rule Modeling Systems[1]

Jan Demey, Mustafa Jarrar, and Robert Meersman[2]

VUB STARLab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels – Belgium
{jdemey, mjarrar, meersman}@vub.ac.be

Abstract. The Internet creates a strong demand for standardized exchange not only of data itself but especially of data semantics, as this same internet increasingly becomes the carrier of e-business activity (e.g. using web services). One way to achieve this is in the form of communicating "rich" conceptual schemas. In this paper we adopt the well-known CM technique of ORM, which has a rich complement of business rule specification, and develop ORM-ML, an XML-based markup language for ORM. Clearly domain modeling of this kind will be closely related to work on so-called ontologies and we will briefly discuss the analogies and differences, introducing methodological patterns for designing distributed business models. Since ORM schemas are typically saved as graphical files, we designed a textual representation as a marked-up document in ORM-ML so we can save these ORM schemas in a more machine exchangeable way that suits networked environments. Moreover, we can now write style sheets to convert such schemas into another syntax, e.g. pseudo natural language, a given rule engine's language, first order logic.

1. Introduction and Motivation

In an enterprise, business rules are used to represent certain aspects of a business domain (static rules) or business policy (dynamic rules). Business rules are defined in [12] as "statements that define or constrain some business aspects. They are intended to assert business or to control or influence its behavior". Modeling such rules is not an easy task since in general it is hard to arrive at the precise understandings and agreements, which they formulate; furthermore these rules may change regularly according to changes in these business aspects. Therefore business rules should be modeled separately in the logical model (i.e. not in the implementation level). They should also be modeled in a declarative manner, in order to enhance their

[1] An early version of this paper has been presented at the "Rule Markup Languages for Business Rules on the Semantic Web" Workshop, 2002.
[2] Author's names are in alphabetical order.

maintainability and reusability [10]. Furthermore, as the volume and spread of networked business enterprises grow, especially in an open environment as the web, business rules play an important role, because agents need to exchange data and transactions according to a shared and agreed set of business rules without misunderstanding. In short, the modeling of business rules should be done at a conceptual level, and in a language that is expressive enough to capture the business complexity [22], but should also be easy and suitable for e.g. business analysts (often non-computer experts) to build and maintain.

Conceptual modeling techniques became especially well known and successful as the basis for graphical CASE tools for building information systems (IS). Many such conceptual modeling techniques exist, for example EER, ORM, the UML, by now often described in classroom textbooks [19][5][15][25]. Conceptual modeling intends to support the quality checks needed before building physical systems by aiming at the representation of data at a high level of abstraction, and therefore acquire a high degree of, often implicit, semantics. This implicitness translated in a requirement for CASE tools to interpret these conceptual specifications representing complex structures and rules (constraints) that must hold on these structures. Also queries and updates may (conceivably) become expressed at a conceptual level, requiring interaction with conceptual structures rather than their implementation (such as relational databases). NIAM [25] is based on an analysis method for natural language, the query and constraint language RIDL [24][16] was developed for this purpose; similarly for ConQuer [3] for NIAM's successor methodology ORM [15].

Using ORM for modeling business rules at a conceptual level has been proposed by e.g. [22][23][14] as a powerful and expressive approach. Indeed, ORM has critical features for this task and it has easy and expressive capabilities in its graphical notation and verbalization possibilities, as will be exploited in the system described in this paper.

In the autonomous, distributed and heterogeneous environment of the internet, there's a strong demand for the exchange of conceptual schemas to be formalized and if possible standardized, since the target application may have a different interpretation or use of a conceptual schema's "surface semantics" (e.g. an EER diagram's topology and linguistic labels, stripped of its geometrical data). Therefore it is very important that the original schema is transcribed (a) as faithfully as possible and (b) in an as standard way as possible, to allow in this way a maximum of flexibility for the target application. Examples of such target applications could be other CASE modelers (even using other meta models), verbalization tools, applications possessing and exploiting "orthogonal" semantics such as spatial, multimedia and temporal databases, or just plain defining formal "semantic" communication protocols as will be needed for the establishment of smart web Services on the so-called Semantic Web [4], [11] operating as the future literally *meaningful* infrastructure for e-commerce and e-business, both at the business to business (B2B) and at the business to customer (B2C) levels.

The format of conceptual models in an ORM CASE tool is usually proprietary and ad-hoc, therefore unsuitable to be exchanged or shared between business agents. Often ORM tools (e.g. Microsoft's VisioModeler®) only generate files for internal use that contain the graphical notation of the conceptual models. Therefore as a

solution we present in this paper an XML-based ORM markup language (ORM-ML, its complete grammar defined in the XML Schema in [13]). This markup language enables exchanging ORM models *including* ORM-syntax business rules.

To facilitate validation for example, or just to provide formal —and consistent— documentation, we also developed a *verbalization style sheet* for ORM-ML documents that allows presenting the facts and the rules in pseudo natural language sentences. Related work on Markup languages for Semantic Web rules can be found in [6] and [20].

We have chosen ORM for its rich constraint vocabulary and well-defined semantics (as did e.g. [9] in an earlier paper) and to use XML Schema to define this communication "protocol" for conceptual schemas seen as XML document instances (for their syntaxes, see [26] [27]). In doing this we chose to respect the ORM structure as much as possible by *not* "collapsing" it first through the usual relational transformer that comes with most ORM-based tools (or UML, or EER tools for that matter —after all, these tools were all conceived mainly to build database schemas for in-house use…).

It is fundamental as well as illustrative of our approach to emphasize the distinction between ORM-ML —subject of this paper— and the related and interesting work that has been reported in [9] using ORM to design XML document *instances*, i.e. which contain instance data described in XML using XML Schema language. In fact, similar to the "classical" use of ORM to generate a relational database schema, in [9] a method for using ORM to design XML Schemas is described, which by definition allows any XML document which may contain such data to be validated against the generated XML Schema. In other words, the syntax of the data in the XML document is modeled using ORM, but this syntax (in this case an XML Schema) is no longer that of an ORM model.

On the other hand, in our approach ORM-ML represents ORM models textually, and the syntax of the resulting model is marked-up by XML tags' syntax (i.e. XML-based structured text document). Therefore the content of this XML document is exactly equivalent to the input ORM model, except for the geometrical information (e.g. shapes, and its positions). The latter could be considered as graphical information of an ORM *diagram*. We therefore defined an XML Schema that can act as a grammar to any ORM-ML document, see Section 3. For the benefits of doing so, see Section 2. In short, the distinction between ORM-ML, and using ORM to generate XML Schema, is that the output document in ORM-ML is a text representation of the ORM model itself, while in the earlier approach the output document is a *transformation* from ORM model to an XML Schema *instance*, which is no longer "ORM".

As a further clarification one might consider the ORM Meta schema in Section 4: its populations are ORM schemas, which our algorithm transforms into ORM-marked-up XML documents. If one would map this same meta schema through the algorithm of [9], the output XML Schema will be a close cousin to our XML Schema in [13]. Comparing the appendix with the example of [9], note also that in our approach the ORM diagram's linguistic elements (names of LOTs, NOLOTs, etc.) stay at the level of string values, emphasizing their flexible instance status while for Bird, Halpin et al. these names become XML tag names, reducing the flexibility by "freezing" them in the generated XML model.

Structure of the Paper: In § 2, we discuss modeling patterns and principles of ontologies, and differences between business rules and ontology rules. § 3 gives a bird's eye introduction to ORM, emphasizing distinguished features of ORM for business rules modeling. § 4 presents ORM-ML and includes a note about the verbalization of ORM-ML files. An algorithm to construct an ORM-ML file from an ORM schema instance (stored in the ORM Meta Schema) is presented in § 5. In § 6 we draw some conclusions including a discussion on some of the perceived advantages of a conceptual schema markup language.

2. Business Rules, Ontology Rules, and the Semantic Web.

The conceptual modeling of a business' domain knowledge using entities, concepts, objects… and their associated events and governing rules has typically always been performed "individually" for the purpose of a given business' application and needs. Modeling domain knowledge "independently" of its application is the subject of the emerging theory and practice of so-called *ontologies*, but it stands to reason that some of the underlying principles and techniques must be in common. We therefore briefly clarify some of the distinctions between the modeling levels of business rules vs. ontology rules for environments like the Semantic Web [4]. An *ontology* in its most general definition is a set of (usually intensional) logical axioms that want to specify or approximate a conceptualization of a certain (e.g. business) domain. Such logical axioms are rules that therefore, typically at the type level, constrain the intended meaning (interpretation) of certain aspects of reality. By representing the semantics in a formal way, agents can share and commit to them, in order to interoperate to exchange data and transactions without misunderstanding. Note that ontologies are somewhat more "subtle" knowledge representations than an information system's data model, which always is "purely" at the type level. In particular they should be sharable, viz. IS-instance (application) independent, but may also refer to relevant "instance concepts" such as "'Euro', 'MasterCard', 'Belgium', etc. Ontologies are more than a mere taxonomy of concepts, since they may contain richer relationships such as "partOf', 'shippedVia, 'OrderedBy', etc.".

As defined in § 1, business rules are intended to constrain or represent a certain aspect of a business domain or policy, thus a similarity also appears between ontology rules and business rules. But notice that business rules will be changed according to the business policy, which changes regularly, and which mostly belongs to one or a few number of enterprises, while ontology rules are more generic and thus more stable, and are intended to be shared by a large number of applications. In short and as a methodological pattern, ontology rules represent a higher level of abstraction than business rules which themselves are on a higher level than the logical and the implementation level.

Linking a business' rules to ontology-based business rules involves aligning (referencing, linking) the concepts and the relationships involved with concepts and relationships of an existing domain ontology. By doing so, the shared understanding

(*semantics*) of the business rules will be improved and thus their reusability will be enhanced as required in open environments such as the (semantic) web.

3. ORM for Modeling Business Rules

In this section we briefly present the modeling principles and capabilities of ORM in terms of modeling requirements for business rules.

ORM was originally intended for modeling and querying databases at a conceptual level where the data requirements of applications need to be represented in a readily understood manner, thus enabling non-IT professionals to assist the modeling, validating, and maintaining processes. ORM offers a number of possibilities for managers, analysts, or domain experts to be involved in the modeling of entity types, domain constraints and business rules by using their own terminology. It is perhaps worthwhile to note that ORM derives from NIAM (Natural Language Information Analysis Method), which was explicitly designed to be a stepwise methodology arriving at "semantics" of a business application's data based on this kind of natural language communication.

ORM has an extensive and powerful graphical notation for representing a domain in a declarative manner as a network of elementary facts and their constraints. Elementary facts are represented in terms of *object types* that play *roles*. This graphical representation can be fairly easily re-verbalized into statements in pseudo natural language in a structured and fixed syntax. Therefore business rule modelers could represent a business policy either graphically or textually or both, which will in general improve, simplify, help to validate, and therefore speed up the modeling process.

Modeling business rules requires an expressive modeling language in order to capture the business complexity. For this, ORM allows representing information structures in multiple ways as unary, binary, as well as n-ary facts. It has a sophisticated object type system that distinguishes between representations of lexical and non-lexical objects, and has strict “is a” relationships with "clean" multiple inheritance as in frame systems ([17]). ORM has an *a priori* given set of static and certain dynamic constraint types and derivation rules that turned out to be suitable and expressive enough to cover a significant part of the needs emerging from enterprise modeling. Such constraints and rules include classical ones such as uniqueness and mandatory roles, as well as less common ones such as subset, equality, ring, derivation and/or stored rules, etc. Rules that do not fit into one of the predetermined rule categories can be formulated using a suitable general-purpose constraint language such as RIDL ([18], [24], [16]).

ORM has well-defined semantics, and the specified facts and constraints can easily be mapped into e.g. first order logic [7]. The finiteness and selection of the set of predetermined constraint types permitted the development of formal validation and consistency analysis tools that check the correctness and the consistency of specified business rules ([8]).

Other advantages include the automated transformation of an ORM business schema into a normalized relational database schema ([8], [15]). This is partially

supported by modern ORM CASE tools such as Microsoft's Visio2000 Architect, VisoModeler, and more fully by the earlier InfoModeler tool [1].

A small Example of an ORM Schema Diagram:

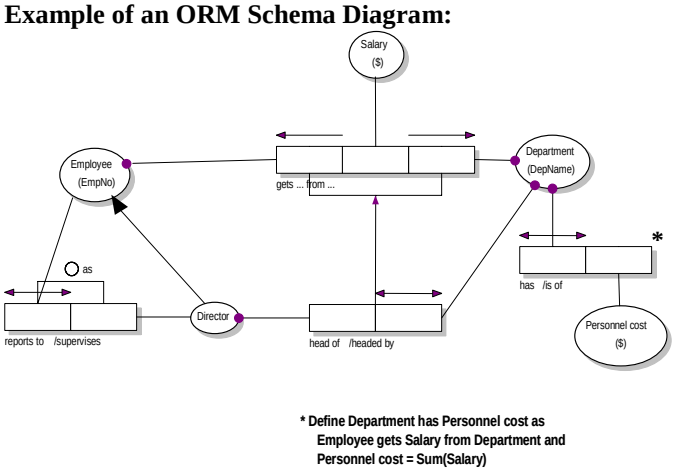


Fig. 1. Example ORM Diagram

On an ORM schema, Object types are shown as named ellipses, with their *reference schemes* in parentheses. Logical predicates (fact types) appear as named sequences of roles, where each role appears as a box. Roles are connected by line segments to the object types that "play" them.

In the Fig. 1, the object types are Employee, Department, Director, Salary and Personnel Cost. Personnel cost and Salary are referenced by an amount of $, Department by a department name (DepName), Employee by an employee number (EmpNo). The arrow connecting the object types Director and Employee denotes an is-a (strict subtype) relationship from Director to Employee. The predicates and subtype link are verbalized as follows: ***Employee reports to Director and Director supervises Employee***; ***Director is_a Employee***; ***Employee gets Salary from Department***; ***Director is head of Department and Department is headed by Director***; ***Department has Personnel cost and Personnel cost is of Department***.

In what follows, we briefly name and explain the constraints occurring in the diagram, in fact by giving an (approximate) verbalization for an example occurring in Fig.1. For other types of ORM constraints, we refer to [24] or [25]; the notation and definitions in this paper will be taken from [15].

Black dots indicate a *mandatory role* constraint. Verbalization in Fig.1: ***each Director is head of at least one Department***. The arrow-tipped bars above the roles are uniqueness constraints. E.g. ***each Department is headed by at most one Director***. Uniqueness constraints can span more than one role, indicating that any combination that instantiates these roles should be unique. E.g. for the predicate ***Employee gets Salary From Department***, there holds ***each Employee gets at most one Salary from a [his] Department***. An arrow between two predicates indicates a subset constraint between the roles involved: ***each Director [who] is head of a Department also works_for that Department***. A circle above a predicate indicates a *ring constraint*. In

the figure the circle marked with ‘as’ indicates an *asymmetric* ring constraint: ***if an Employee reports to a Director*** (who is also an Employee**), *then [his] Director must not report to this Employee***.

Finally, an asterisk beside a predicate indicates that we have a *derived fact type*. The derivation rule is then included elsewhere, linked to the schema. In Fig.1, define ***Personnel cost for a Department*** as sum of all ***Salaries of Employees received_from that [their] Department***. Instances of derived facts may be considered *stored* (i.e. pre-calculated at compile time and maintained by updates) or *interpreted* (i.e. computed on-the-fly when needed).

4. ORM-Markup Language

The ORM conceptual schema methodology is fairly comprehensive in its treatment of many "practical" or "standard" business rules and constraint types. Its detailed formal description, (we shall take ours from [15]) makes it an interesting candidate to non-trivially illustrate our XML based ORM-markup language as an exchange protocol for representing ORM conceptual models. In this section we describe the main elements of the ORM-ML grammar and demonstrate it using a few selected elementary examples. A complete formal definition of the grammar for this ORM-ML as an XML Schema instance can be found in [13]. It follows that an ORM Schema when formulated in ORM-ML must be valid according the defined XML Schema. A more complete example is provided in the appendix.

ORM-ML allows the representation of any ORM schema without loss of information or change in semantics, except for the geometry and topology (graphical layout) of the schema (e.g. location, shapes of the symbols), which we however easily may provide as a separate *graphical style sheet* to the ORM Schema (not added in this paper).

We represent the ORM document as a one node element called ORMSchema, which consists itself of two nodes: ORMMeta and ORMBody. As a header to an ORM document, and to illustrate the "ORM Schema Document" (instance) nature of the described schema, ORMMeta node includes *metadata* about the ORM *document* using the 16 well-known Dublin Core Meta Tags [RFC2431]; an example of their use appears in Table 1 below.

Table 1. Example of an ORMMeta Node in an ORM-ML File

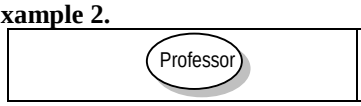
```
…<ORMMeta>
        <dc:title>ORM-ML example</dc:title>
        <dc:creator>Jan Demey</dc:creator>
        <dc:description>A complete example of an ORM-ML file</dc:description>
        <dc:contributor>Mustafa Jarrar</dc:contributor>
        <dc:contributor>Robert Meersman</dc:contributor>
</ORMMeta>….
```

The ORMBody node consists of at most these five different kinds of (meta-ORM) elements: Object, Subtype, Predicate, Predicate_Object and Constraint.

We adopt in the sequel the ORM modeling technique as defined in [15] except for some minor nomenclature and notation differences, argued in more detail

elsewhere, which add some additional abstraction and precision. Object elements are abstract XML elements and are used to represent Object Types. They are identified by an attribute 'Name' which is the name of the Object Type in the ORM Schema (see the figure in Example 2). Objects might have some Value or ValueRange elements, which are used for *value constraints* on the Object Type (not present in Fig.2). A ValueRange element has 2 attributes: *begin* and *end*, with obvious meanings. Objects are implemented by two XML elements: LOT (Lexical Object Type, called Value Types in [15]) and NOLOT (Non-Lexical Object Type, called Entity Types in [15]). LOT elements may have a *numeric* attribute, which is a boolean and indicates whether we deal with a numeric Lexical Object Type. NOLOT elements have a boolean attribute called *independent*, which indicates whether the Non Lexical Object Type is independent (see [15] for definitions). NOLOT elements may also have a *reference* element. A reference element would indicate how this NOLOT is identified by LOTs and other NOLOTs in a given application environment. A reference element has 2 attributes: ref_name, the name of the reference and numeric, a boolean to indicate whether it is a numeric reference.

Example 2.



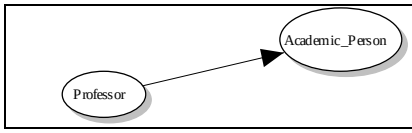

```
…
<Object xsi:type="NOLOT" Name="Professor"/>
…
```

Fig. 2

Table 2. ORM-ML representation of Fig.2

Subtype elements are used to represent subtype relationships between (non-lexical) object types. A subset element is required to have two attributes: parent and child, which are references to object elements (see Example 3).

Example 3.




```
…
<Object xsi:type="NOLOT" Name="Professor"/>
<Object xsi:type="NOLOT" Name="Academic_Person"/>
<Subtype Parent="Academic_Person" Child="Professor">
…
```

Fig.3.

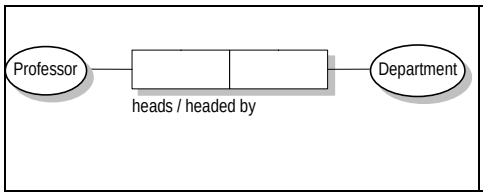
Table 3. ORM-ML representation of Fig.3

Predicates consist of at least one Object_Role element. Such an element contains a reference to an object and may contain a role. They actually represent the rectangles in an ORM schema. Every Object_Role element needs a generated attribute 'ID' which identifies the Object_Role. By using this ID attribute, we can refer to a particular Object_Role element in the rest of the XML document, which we will need to do when e.g. we define constraints.

Predicates can have one or more rule elements. These elements can contain extra rules that are defined for the predicate.

Predicates also have two boolean attributes that are optional: '*Derived*' and '*Derived_Stored*' which indicate whether a predicate respectively is derived, or derived and stored, or not.

Example 4. This example shows a simple binary predicate as in fig 4, and how it is represented in ORM-ML in Table 4.



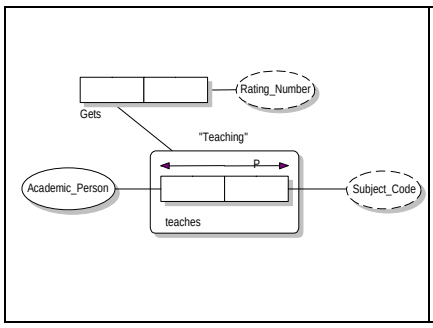

```
…
<Object xsi:type="NOLOT" Name="Professor"/>
<Object xsi:type="NOLOT" Name="Department"/>
<Predicate>
    <Object_Role ID="ID1" Object=”Professor”
        Role=”heads”/>
    <Object_Role ID="ID2" Object=”Department”
        Role=”headed_by”/>
</Predicate>
…
```

Fig.4.

Table 4. ORM-ML representation of Fig.4

Predicate_Objects are actually objectified predicates, which are used in nested fact types. They contain a predicate element and have an attribute called 'Predicate_Name'. So in fact they are merely a predicate that has received a (new) object type name. In building Object_Roles, the Predicate_Name can be referenced. In this way we build predicates that contain objectified predicates instead of object types. Example 5 illustrates the XML representation for nested fact types that this requires.

Example 5.
This example shows the representation of a nested fact type as in Fig. 5.




```
…
<Object xsi:type="NOLOT" Name="Academic_Person"/>
<Object xsi:type="LOT" Name="Subject_Code"/>
<Object xsi:type="LOT" Name="Rating_Number"/>
<Predicate_Object Predicate_Name=”Teaching”>
  <Predicate>
    <Object_Role ID="ID18" Object=”Academic_Person”
        Role=”teaches”/>
    <Object_Role ID="ID19" Object=”Subject_Code”/>
  </Predicate>
</Predicate_Object>
<Predicate>
  <Object_Role ID="ID18" Object="Teaching"
      Role="Gets"/>
  <Object_Role ID="ID19" Object="Rating_Number"/>
</Predicate>
…
```

Fig.5.

Table 5. ORM-ML representation of Fig.5

Constraint elements represent the ORM constraints. The Constraint element itself is abstract, but it is implemented by different types of constraints, viz. mandatory, uniqueness, subset, equality, exclusion, frequency, irreflexive, anti-symmetric, asymmetric, symmetric, intransitive, and acyclic constraints. As mentioned above, we use the ID-s of the Object_Role elements to define constraints (except for value

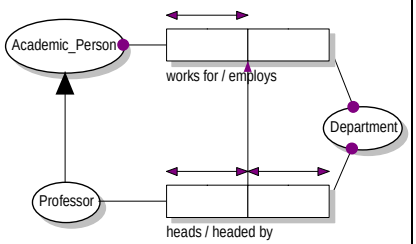
constraints on an object type, because these constraints are defined in the corresponding object element).

Uniqueness and mandatory constraint elements possess only Object_Role elements (at least one). These elements are the object_roles in the ORM diagram on which the constraint is placed. In this way, there is no need to make a distinction between the ORM-ML syntax of "external" and "internal" uniqueness constraints (see [15]), or between mandatory and disjunctive mandatory constraints (see Example 6 below).

The representation for subset, equality and exclusion constraints is analogous, so we will only discuss them in general terms. Each of these latter constraints has exactly two elements that contain references to (combinations of) object_role elements. For instance, to represent an equality constraint between two predicates, we create a subset element, containing 2 elements ‘First’ and ‘Second’. In the first element we put references to the object_roles from the first predicate, and in the second we put references to the object_roles from the second predicate (see Example 6).

Example 6.

This example shows the representation of the constraints from Fig. 6.




```
<Predicate>
  <Object_Role ID="ID8" Object="Academic_Person"
      Role="WorksFor"/>
  <Object_Role ID="ID9" Object="Department"
      Role="Employs"/>
</Predicate>
<Predicate>
  <Object_Role ID="ID12" Object="Professor"
      Role="Heads"/>
  <Object_Role ID="ID13" Object="Department"
      Role="Headed by"/>
</Predicate>
<Constraint xsi:type="Mandatory">
  <Object_Role>ID8</Object_Role>
</Constraint>
<Constraint xsi:type="Uniqueness">
  <Object_Role>ID8</Object_Role>
</Constraint>
<Constraint xsi:type="Uniqueness">
  <Object_Role>ID12</Object_Role>
</Constraint>
<Constraint xsi:type="Uniqueness">
  <Object_Role>ID13</Object_Role>
</Constraint>
<Constraint xsi:type="Mandatory">
  <Object_Role>ID9</Object_Role>
</Constraint>
<Constraint xsi:type="Mandatory">
  <Object_Role>ID13</Object_Role>
</Constraint>
<Constraint xsi:type="Subset">
  <Parent>
    <Object_Role>ID8</Object_Role>
    <Object_Role>ID9</Object_Role>
  </Parent>
  <Child>
    <Object_Role>ID12</Object_Role>
    <Object_Role>ID13</Object_Role>
  </Child> </Constraint>
```

Fig.6.

Table 6 ORM-ML representation of Fig.6

Finally, *ring constraint* elements simply contain references to the object_roles they are put on, and frequency constraints have two attributes: a reference to the object_role the constraint is placed on and an attribute called ‘Frequency’ which contains the declared frequency number.

A Note on Verbalization Style Sheets for Business Rules. Verbalization of a conceptual model is the process of writing its facts and constraints in pseudo natural language sentences, which assumedly allows non-experts to (help) check, validate, or even build conceptual schemas. The ORM modeling tool “InfoModeler” supported a built-in feature for automatic verbalization of ORM Schema or part of it. In ORM-ML, generating such verbalizations from agreed templates (i.e. "template NL" syntax) parameterized over the ORM schema is done by building separate XML-based style sheets. Moreover, multilingual style sheets[3] also become easier by translating these template sentences into different languages, its parameter values (which come from the ORM schema) translated by a human or machine.

5. Generating an ORM-ML file from an ORM Schema

XML being a computer-friendly language, it is of course not the ultimate purpose to write ORM-ML files by hand. Although it turns out relatively easy to do that, the goal must be to implement into existing conceptual modeling tools, ideally, a functionality like a “Save/Load as ORM-ML” dialog box. Because in general the repository format in which ORM (or other modeling method's) schemas are stored is proprietary or even "closed" inside the CASE tool's software, we will here just show in abstract terms the algorithm how to make the conversion, starting from a rather simplified *meta schema* for ORM given below (in ORM diagram itself, of course) in Figure 7. Remember that by the definition of Meta schema, individual ORM schema instances are considered to be stored "conceptually" in such a Meta schema in an obvious manner. After the customary application of a conceptual-to-relational transformation algorithm, its actual content is retrieved from the relational database of which the relational database tables were derived from this Meta schema (see [15], for theory and examples of this).

[3] For ORM-ML, e.g. a multilingual verbalization style sheet was constructed in the authors’ lab [ORMML], based on the XML Schema in [13], (but not discussed in this paper).

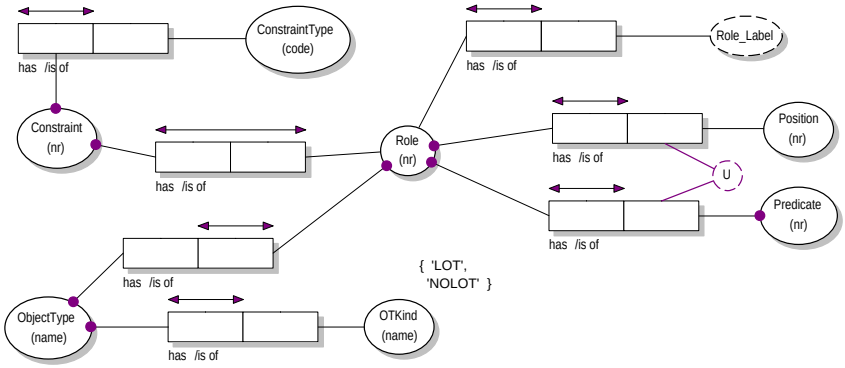


Fig.7.

Suppose we store instances of this Meta schema in a relational database with the tables shown in figure 8.

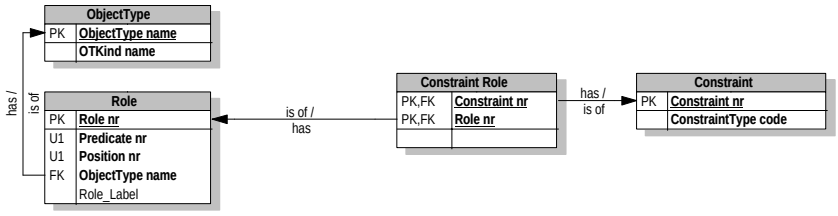


Fig.8.

Now we easily convert its contents to ORM-ML, for instance as follows (it will be useful to consult the appendix for a detailed illustration of its result):

Sketch of ORM to ORM-ML mapping algorithm
(a) Obtain all *ObjectType* elements by a query **SELECT * FROM ObjectType**. Each returned row will be one ObjectType, with as name attribute ObjectType_name and as xsi:type attribute OTKind.

(b) Get predicates by issuing a query **SELECT Role_nr, Predicate_nr, Object_Type_name, Role_Label FROM Role ORDER BY Predicate_nr, Position_nr**. Start a new *predicate* element, and append an *Object_Role* element (attributes ID: Role_nr, Object: Object_Type_name, Role: Role_Label) for each row returned until the Predicate_nr changes. When the Predicate_nr changes, close the predicate element and start a new one. Repeat this until all returned rows are processed.

(c) For *Constraint* elements issue a query **SELECT C.Constraint_nr, C.ConstraintType_code, CR.Role_nr from Constraint C, Constraint_Role CR WHERE C.Constraint_nr = CR.Constraint_nr**. Open a Constraint element

(attribute type: C.ConstraintType_code). Append an Object_Role element (content: CR.Role_nr) for each row returned until C.Constraint_nr changes. When C.Constraint_nr changes, close the Constraint element and start a new one until all returned rows are processed.

Note. Naturally one could specify the above algorithm also by directly "querying" the ORM Meta Schema in Fig. 7 itself, using a suitable ORM-based query language such as RIDL [18] instead of SQL. Although this would lead to a more compact (and "conceptual" specification, we lacked the space in this paper to present the (original) RIDL language, which may be downloaded from http://www.starlab.vub.ac.be/…-> RIDL_User_Guid.zip

6. Conclusion: some other advantages for ORM-ML, and future research

In this paper we have explained how ORM could be used to design business rules. We have presented a way to save ORM schemas in an XML-based markup language. The main advantage of this markup language is that it is easy to exchange information. Like ORM-ML, any conceptual modeling approach could have a markup language, since by standardizing such a markup language several advantages are worth noting.

- *Helps schema integration and transformation.*

In information systems, it is in general easier to integrate or *align* the conceptual models of the systems than to materially integrate the logical or the physical internal representation of the system, as demonstrated by the literature on view- and schema integration (e.g. [21]). Therefore, ORM-ML as a standardized syntax for ORM models may assist interoperation tools to exchange, parse or understand the ORM schemas.

- *Interoperability for exchanging and sharing conceptual models over the Internet.*

Facilities are needed to share and exchange ORM conceptual models (not only business rules) in terms of a networked, distributed computing-driven, and collaborative environment, and to allow users to browse and edit shared domain knowledge over the Internet, intranets and other channels. A conceptual schema markup language provides a standardizable method to achieve interoperability among CASE tools that use that conceptual modeling technique.

- *Implementing a conceptual query language over the Web.*

In open and distributed environments, building queries should be possible regardless of the internal representation of the data. Query languages based on ontologies (seen as shared conceptual models) help users not only to build queries, but also make them easier, more expressive, and more understandable than corresponding queries in a language like SQL. Exchanging, reusing, or

sharing such queries efficiently between agents over the web is substantially facilitated by a standardized markup language. Consequently, ORM-based query languages (e.g. RIDL [24], [18], ConQuer [3]) would gain from ORM-ML by representing queries in such an exchangeable representation.

- *Building a standard style sheet to generate its formalizations.*

Generating verbalization style sheets for a given usage or need may require a certain kind of style sheets e.g. for first order rewriting formalisms of ORM-ML documents, or to transform the XML-based representation into another XML-based representation. Another important and strategic issue is that one could write a style sheet to generate the given ORM model instance into a given rule-engine's syntax, which allows run-time interpretation by that rule engine, for instance performing instance validation, integrity checks, etc. This also enhances the use of ORM-ML for representing and modeling ORM-based business rules at a conceptual level, making agents able to share, reuse, and exchange these rules.

Finally, it is clearly also possible to transform ORM-ML into another language's syntax. We illustrated in passing how ORM-ML may be transformed into/from (structured) pseudo natural language. Currently we are e.g. implementing and investigating mappings from ORM-inspired ontologies into languages used by a number of commercial (e.g. Haley's Authorete® [2]) and open-source rule engines.

Acknowledgments. Partial support for the reported work from the EC FP5 Thematic Network OntoWeb (IST-2000-29243) is hereby also gratefully acknowledged.

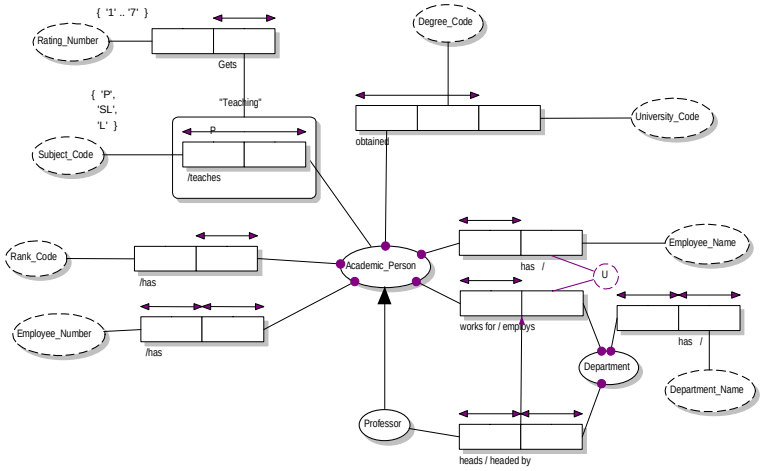
References

1. Asymetrix. InfoModeler User Manual. Asymetrix Corporation, 110-110th Avenue NE, Suite 700,Bellevue, WA 98004, Washington (1994)
2. http://www.haley.com/Authorete.html
3. Bloesch, A., Halpin, T.: Conquer: A Conceptual Query Language. In: Thalheim, B. (ed.): Conceptual Modeling - ER'96 Proceedings. LNCS, Springer Verlag (1996)
4. Berners-Lee, T. et al.: Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor. Harper San Francisco (1999)
5. Booch, G., Rumbaugh, J., Jacobson, I.: The UML User Guide. Addison-Wesley (1999)
6. Boley, H., Tabet, S., Wagner, G.: Design Rationale of 20: A Markup Language for Semantic Web Rules. In: International Semantic Web Working Symposium (SWWS) (2001)
7. De Troyer, O., Meersman, R.: A Logic Framework for a Semantics of Object-Oriented Data Modeling. OOER (1995) 238-249
8. De Troyer, O., Meersman, R.A., Verlinder, P.: RIDL* on the CRIS Case: a Workbench for NIAM. In: Olle, T.W., Verrijn-Stuart, A.A., Bhabuta, L.

(eds.): Computerized Assistance during the Information Systems Life Cyle. Elsevier Science Publishers B.V. North-Holland (1988) 375-459
9. Bird, L., Goodchild, A., Halpin, T.A.: Object Role Modelling and XML-Schema. In: Laender, A., Liddle, S., Storey, V. (eds.): Proceedings of the 19th International Conference on Conceptual Modeling (ER'00). LNCS, Springer Verlag (1999)
10. Gottesdiener, E.: Business RULES show power, Promise. In: Issue of Application Development Trends. vol4. no3 (1997)
11. Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. Technical Report, Vrije Universiteit Amsterdam
12. Guide Business Rules Project, 11/95, GUIDE International Corporation, available at http://www.guide.org/ap/apbrules.htm. In Business RULES show power, Promise, by E. Gottesdiener, President, EBG consulting, Issue of application Development Trends, vol4., no3, (1997)
13. http://www.starlab.vub.ac.be/ORMML/ormml.xsd
14. North, K.: Modeling, Data Semantics, and Natural Language. In: New Architect magazine (1999)
15. Halpin, T.: Information Modeling and Relational Databases. 3rd Ed., Morgan-Kaufmann (2001)
16. NN.: Terminology for the Conceptual Schema and Information System. ISO Technical Report TR9007, ISO (1990)
17. Karp, P.D.: The design space of frame knowledge representation systems, Technical Report 520, SRI International AI Center (1992)
18. Meersman, R.: Languages for the High-Level End User. In: InfoTech State of the Art Report, Pergamon Press (1981)
19. Navathe, S., Elmasri, R.: Fundamentals of Database Systems, 3rd ed., Addison-Wesley (2001)
20. http://www.dfki.uni-kl.de/20/
21. Spaccapietra, S., Parent, C.: View Integration: A Step Forward in Solving Structural Conflicts, IEEE Transactions on Data and Knowledge Engineering 6(2), IEEE (1994)
22. Halpin, T.: Business Rules and Object Role Modeling. In: Issue of Database Programming & Design, vol. 9, no. 10 (1996) 66-72
23. Halpin, T.: Modeling for Data and Business Rules. An interview with Terry Halpin conducted by Ron Ross, Editor of Database Newsletter. In: Issue of the Database Newsletter (1997)
24. Verheyen, G., van Bekkum, P.: NIAM, An Information Analysis Method. In: Olle, T.W., Sol, H., Verrijn-Stuart, A. (eds.): IFIP Conference on Comparative Review of Information Systems Methodologies, North-Holland (1982).
25. Wintraecken, J.J.V.R.: The NIAM Information Analysis Method: Theory and Practice. Kluwer Deventer Netherlands (1990)
26. http://www.w3.org/XML/
27. http://www.w3.org/XML/Schema

Appendix

A complete example of an ORM Schema with the associated ORM-ML file generated by the algorithm defined in Section 5.




```
<?xml version="1.0" encoding="UTF-8"?>
<ORMSchema xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation=http://starlab.vub.ac.be/ORMML/ormml.xsd
xmlns:dc="http://purl.org/dc/elements/1.1/">
        <ORMMeta>
                <dc:title>ORM ML example</dc:title>
                <dc:creator>Jan Demey</dc:creator>
                <dc:description>A complete example of an ORM ML file</dc:description>
                <dc:contributor>Mustafa Jarrar</dc:contributor>
                <dc:contributor>Robert Meersman</dc:contributor>
        </ORMMeta>
        <ORMBody>
                <Object xsi:type="NOLOT" Name="Academic_Person"/>
                <Object xsi:type="LOT" Name="Employee_Number"/>
                <Object xsi:type="LOT" Name="Degree_Code"/>
                <Object xsi:type="LOT" Name="University_Code"/>
                <Object xsi:type="LOT" Name="Employee_Name"/>
                <Object xsi:type="NOLOT" Name="Professor"/>
                <Object xsi:type="NOLOT" Name="Department"/>
                <Object xsi:type="LOT" Name="Department_Name"/>
                <Object xsi:type="LOT" Name="Rating_Number">
                        <ValueRange begin="1" end="7"/>
                </Object>
                <Object xsi:type="LOT" Name="Subject_Code">
                        <Value>P</Value>
                        <Value>SL</Value>
                        <Value>L</Value>
                </Object>
                <Object xsi:type="LOT" Name="Rank_Code"/>
                <Subtype Parent="Academic_Person" Child="Professor"/>
                <Predicate>
                        <Object_Role ID="ID1" Object="Academic_Person" Role="Has"/>
                        <Object_Role ID="ID2" Object="Employee_Number"/>
                </Predicate>
                <Predicate>
                        <Object_Role ID="ID3" Object="Academic_Person" Role="Obtained"/>
                        <Object_Role ID="ID4" Object="Degree_Code"/>
                        <Object_Role ID="ID5" Object="University_Code"/>
                </Predicate>
                <Predicate>
                        <Object_Role ID="ID6" Object="Academic_Person" Role="Has"/>
                        <Object_Role ID="ID7" Object="Employee_Name"/>
                </Predicate>
                <Predicate>
                        <Object_Role ID="ID8" Object="Academic_Person" Role="WorksFor"/>
                        <Object_Role ID="ID9" Object="Department" Role=”Employs”/>
                </Predicate>
                <Predicate>
                        <Object_Role ID="ID10" Object="Department" Role="has"/>
                        <Object_Role ID="ID11" Object="Department_Name"/>
                </Predicate>
                <Predicate>
```

```
                        <Object_Role ID="ID12" Object="Professor" Role="Heads"/>
                        <Object_Role ID="ID13" Object="Department" Role="Headed by"/>
                </Predicate>
                <Predicate>
                        <Object_Role ID="ID14" Object="Academic_Person" Role="Has"/>
                        <Object_Role ID="ID15" Object="Rank_Code"/>
                </Predicate>
                <Predicate>
                        <Object_Role ID="ID18" Object="Teaching" Role="Gets"/>
                        <Object_Role ID="ID19" Object="Rating_Number"/>
                </Predicate>
                <Predicate_Object Predicate_Name="Teaching">
                        <Predicate>
                                <Object_Role ID="ID16" Object="Academic_Person" Role="Teaches"/>
                                <Object_Role ID="ID17" Object="Subject_Code"/>
                        </Predicate>
                </Predicate_Object>
                <Constraint xsi:type="Uniqueness">
                        <Object_Role>ID1</Object_Role>
                </Constraint>
                <Constraint xsi:type="Mandatory">
                        <Object_Role>ID1</Object_Role>
                </Constraint>
                <Constraint xsi:type="Uniqueness">
                        <Object_Role>ID2</Object_Role>
                </Constraint>
                <Constraint xsi:type="Mandatory">
                        <Object_Role>ID3</Object_Role>
                </Constraint>
                <Constraint xsi:type="Uniqueness">
                        <Object_Role>ID3</Object_Role>
                        <Object_Role>ID4</Object_Role>
                </Constraint>
                <Constraint xsi:type="Mandatory">
                        <Object_Role>ID6</Object_Role>
                </Constraint>
                <Constraint xsi:type="Uniqueness">
                        <Object_Role>ID6</Object_Role>
                </Constraint>
                <Constraint xsi:type="Mandatory">
                        <Object_Role>ID8</Object_Role>
                </Constraint>
                <Constraint xsi:type="Uniqueness">
                        <Object_Role>ID8</Object_Role>
                </Constraint>
                <Constraint xsi:type="Uniqueness">
                        <Object_Role>ID12</Object_Role>
                </Constraint>
                <Constraint xsi:type="Uniqueness">
                        <Object_Role>ID13</Object_Role>
                </Constraint>
                <Constraint xsi:type="Uniqueness">
                        <Object_Role>ID10</Object_Role>
                </Constraint>
                <Constraint xsi:type="Mandatory">
                        <Object_Role>ID10</Object_Role>
                </Constraint>
                <Constraint xsi:type="Mandatory">
                        <Object_Role>ID9</Object_Role>
                </Constraint>
                <Constraint xsi:type="Mandatory">
                        <Object_Role>ID13</Object_Role>
                </Constraint>
                <Constraint xsi:type="Uniqueness">
                        <Object_Role>ID11</Object_Role>
                </Constraint>
                <Constraint xsi:type="Uniqueness">
                        <Object_Role>ID7</Object_Role>
                        <Object_Role>ID9</Object_Role>
                </Constraint>
                <Constraint xsi:type="Subset">
                        <Parent>
                                <Object_Role>ID8</Object_Role>
                                <Object_Role>ID9</Object_Role>
                        </Parent>
                        <Child>
                                <Object_Role>ID12</Object_Role>
                                <Object_Role>ID13</Object_Role>
                        </Child>
                </Constraint>
                <Constraint xsi:type="Mandatory">
                        <Object_Role>ID14</Object_Role>
                </Constraint>
                <Constraint xsi:type="Uniqueness">
                        <Object_Role>ID14</Object_Role>
                </Constraint>
                <Constraint xsi:type="Uniqueness">
                        <Object_Role>ID16</Object_Role>
                        <Object_Role>ID17</Object_Role>
                </Constraint>
                <Constraint xsi:type="Uniqueness">
                        <Object_Role>ID18</Object_Role>
                </Constraint>
        </ORMBody>
</ORMSchema>
```