**ORM-ODM: Ontology Definition Metamodel for Object Role Modeling**

PAN Wen-lin  
College of Mechanical and Electrical Engineering  
Harbin Engineering University  
Harbin 150001, China  
panwenlin@hrbeu.edu.cn

LIU Da-xin  
College of Computer Science and Technology  
Harbin Engineering University  
Harbin 150001, China  
liudaxin@hrbeu.edu.cn

**Abstract**—In order to share, exchange, and process ontologies and business rules modeling by Object-Role Modeling (ORM) on Internet and the open connectivity environments, it is necessary to develop a standard ORM meta-model by ontology meta-modeling method based on MOF 2.0. A MOF-compliant ontology definition meta-model for ORM 2.0 (ORM-ODM) was proposed and validated by MOF-supported tools.

**Keywords**—Object Role Modeling; ORM; Ontology Definition Metamodel; ODM; ORM meta-model

I. INTRODUCTION

ORM (Object Role Modeling), current version is 2.0, is a conceptual modeling approach that was developed in the early 1970s based on the NIAM (Natural Information Analysis Method) [1]. Compare to other class-attribute conceptual modeling methods, such as ER and UML, ORM is an attribute-free approach that can build more stable models and represent more complex business rules. ORM graphical models can be translated into pseudo natural language using mix-fix predicates of any arity that make it easier, also for domain experts, to create, check and adapt the knowledge about the UoD (Universe of Domain).

Today, ORM has been used in ontology engineering to model domain ontologies, such as DOGMA [2]; Object Management Group (OMG) has adopted ORM to represent business rules in SBVR (Semantics of Business Vocabulary and Business Rules) specification [3].

In order to share, interchange, and process ontologies and business rules modeling by ORM on Internet and the open connectivity environments, it is necessary to develop a standard MOF-compliant ORM meta-model.

The Meta Object Facility (MOF), an adopted OMG standard, provides a meta-model management framework and a set of meta-model services to enable the development and interoperability of model and metadata driven systems [4]. MOF 2.0 can be used to define and integrate a family of meta-models using simple class modeling concepts. Based on MOF and XMI (XML Metadata Interchange) [5], OMG proposed a standard called Ontology Definition Metamodel (ODM) to provide a series of standard meta-model for RDF (Resource Description Framework), OWL (Web Ontology Language), DL (Description Logic), CL (Common Logic), ER (Entity Relationship), etc [6]. However, this standard did not provide ODM for ORM.

Terry Halpin has proposed a meta-model defined by ORM [7]; Jarrar, et al proposed a XML markup language for ORM called ORM-ML [8-9], it is also an ORM meta-model defined by XML schema. Cuyler and Halpin proposed two ORM meta-model modeling by ORM and UML, and discussed a working prototype that is compliant with MOF [10]. All of these ORM meta-model, no one can fully meet the MOF standard. Based on above works, this paper proposed a complete and MOF-compliant meta-model for ORM 2.0 called ORM-ODM. This ORM meta-model was validated and an appropriate MOF XMI file was generated through MagicDraw UML.

II. ORM MODEL/SHEMA

An ORM model or schema is either documented in a XML file or stored in a database. An ORM model consists of two parts: the first one is description about the model, such as “Title”, “Creator”, “Date”, “Version”, etc; the second one is ORM model data. ORM Schema and its two parts can be modeled with three classes: Schema, Head and Body, as shown in figure 1.

![Class diagram for ORM Schema](image)

Figure 1. Class diagram for ORM Schema

To facilitate the management and maintenance for ORM model, it is necessary to document description information about the model. Head represents description about ORM model; it is composed by one or more description. We use Info class to modeling description information, and two attributes, name and value, to represent the type and content of that information. For example, name is “Author” and value is “PAN Wen-lin”.

Body is the main part of ORM model which composed by at least one type of model data, such as object and/or fact, etc. The ORM model data can be classified into four types:
object type, subtype/supertype relationship, predicate or fact type, and constraint. Four classes, ObjectType, FactType, Subtyping, and Constraint, are used to represent these four types; ObjectType and Constraint are all abstract types, can be partitioned into many concrete and/or abstract subtypes.

When the Universe of Discourse (UoD), i.e. business domain, is too complex to manage, it needs to divide the UoD into some sub-sections, and this process can iterate until each sub-section is manageable. When each sub-section’s ORM model, called sub-schema, was established, it need to integrate the sub-schemas into an upper or global conceptual schema. An upper schema can compose by one or more independent sub-schemas plus an integrated model. For that reason, we use class SubSchema to represent sub-schema and it is also a part of class Body. Because each schema/sub-schema is stored in an XML file or database, URI attribute is used to refer a sub-schema.

III. OBJECT TYPE AND SUBTYPING

Figure 2 shows the class diagram of object type and subtyping relationship between object types.

We use two attributes, refName and isNumeric, to model this feature of ORM. The refName attribute represents the identifier to refer, such as “name”, “code”, and so on; and isNumeric attribute represent the value type of that identifier, when the value of isNumeric is true means that the value of refName is a numeric, otherwise it is a string.

An entity type object is either atomic or nested, i.e. EntityType is an abstract class and it has two partitioned subclasses: AtomicEntityType and NestedEntityType. Atomic entity type is primitive entity type. Nested entity type is objectified from fact type which discussed in next section.

In ORM schema, some entity types are referred from other ORM schemas, these entity types are all external; it needs to point out the URI of these external schemas in order to locate the external entities. We use ExternalEntityType class with an attribute schemaURI to model this feature. ExternalEntityType is also a subclass of EntityType.

B. Subtyping Relationship Between Two Object Types

There exists a generalization/specialization relationship between two object types called subtyping. In this relationship, one object type plays super-type role, another play sub-type role. Some subtype relationships are declared directly, others are defined by a derivation rule or expression. ORM support multi-inherence, if an object has more than one super-type, only one is primary inheritance. For the sake of indexing, every subtype relationship must have an identifier to distinguish with others.

We use Subtyping class to model subtype relationship between two objects. Subtyping has four attributes, ID, isPrimary, derivationRule, and subRoleName, to represent features of subtyping relationship mentioned above. ID represents the identifier; isPrimary represents whether the subtyping is a primary inheritance; subRoleName represents the name of subtype role; and derivationRule represents the expression used to define the subtyping relationship.

An expression is a formatted string expressed by a specified rule language, such as KL, OCL, etc. We use Expression class and two attributes language and body to model this feature. The language attribute represents the rule language used by expression. The body attribute represents the content of an expression.

IV. PREDICATE/FACT TYPE AND OBJECTIFICATION

Fact type and predicate have same semantic in ORM. A fact is a proposition that is taken to be true by the relevant business community, where the proposition is elementary or existential. A Fact type/predicate is a declarative sentence with object holes in it. To complete the sentence, the object holes or placeholders are filled in by object terms (i.e. object’s name). An n-ary predicate is a sentence with n object holes (n > 0). Since the order is significant, a filled-in n-ary predicate is associated with a sequence of n object terms, not necessarily distinct. Each object hole in a predicate is a role played by an object.

Objectification or nesting is a kind of schema transformation to making an object out of a relationship. An object formed by objectification is called an objectified relationship. The type of object so formed is called an objectified
association, an objectification type, or a nested entity type. A nested entity type is also an object type.

Figure 3 shows the class diagram of fact type and its objectification feature.

We use class FactType and an identifier attribute ID to model fact type. FactType is composed by one or more ObjectRoles and PredicateTexts in a special order. PredicateText represents the text parts of a sentence; text attribute represents a text part. Each ObjectRole only played by one ObjectType; each ObjectType can play many ObjectRoles. If ObjectType is independent, it does not play any ObjectRole. ID attribute is the identifier of ObjectType, and roleName attribute is used to represent the name of object role.

Although the order of ObjectRole and PredicateText can identify their stored order, use a numeric attribute seqNumber to identify this order can sure the correct composed order in any case.

A FactType can objectify as a NestedEntityType; a NestedEntityType must objectified by a FactType.

Some fact types are derived or semi-derived from others by some rules; such fact type is called as derived fact type. Some derived facts need persistence, and others need not. We use DerivedFactType class to model a derived fact type; the derivationRule attribute represents the derived rule, it is an Expression; the logic attribute stored represents whether derived facts need to store; and the logic attribute semi-derived represents only part facts are derived.

V. CONSTRAINTS

ORM constraints can be partitioned into following types: constraints on subtype relationship, value constraints on objects and/or object roles, constraints on object roles (such as mandatory or XOR (Exclusive and OR), internal/external uniqueness and value-compare), set constraints on set of object-roles (such as subset, equality and exclusion), occur frequency or cardinality constraints, ring constraints, and textual constraints. Figure 4 shows the class diagram of all types of ORM constraints. Constraint class and its subclasses can represent ORM constraints and its sub-types.

Constraint is an abstract class; ID attribute is its identifier. Each ORM constraint has two modalities: deontic or alethic, deontic attribute is used to identify whether constraint is in deontic modality; when the value of deontic is true means the constraint is deontic, otherwise it is alethic.

A. Textual Constraint

Textual constraint is a rule expression or a purely text to describe business rules which can not be expressed by graphic constraint annotations. TextualConstraint class is used to model textual constraint; the rule attribute represents rule expression; the text attribute represents a purely text to describe business rule which can’t express by an expression.

B. Constraint on Subtyping

There are three kinds of constraints related to two or more subtyping relationships: exclusive, total, and partition. Partition is equal to exclusive plus total. Figure 5 shows the class diagram of subtype constraint. SubtypingConstraint class is used to model this constraint type. Because not all subtyping must restrict by this constraint, the multiplicity on constraint end is “0..*”; another end is “2..*”.

Two logical attributes isTotal and isExclusive are used to represent three kinds of constraint: the value of isTotal is true means it is a total constraint; the value of isExclusive is true means it is an exclusive constraint; the value of isTotal and isExclusive are both true means it is a partition constraint.

C. Value Constraints on Object and Role

Figure 6 shows the class diagram of value constraint. ValueConstraint class is used to represent this constraint type,
it is an abstract class. Two partitioned subclasses are ObjectValueConstraint and RoleValueConstraint, restrict on object and role respectively. At most one value constraint can restrict on one object/role; each value constraint is restricted on exactly one object/role.

![Diagram of Value Constraint](image)

**Figure 6.** Class diagram of value constraint on object/role

The value constraint has three forms whether it restrict on an object or a role: enumerated literal, semi-bounded discrete range, and/or bounded range (include discrete range and bounded continues range). A value constraint may be one of three forms or combination of them, i.e. a value constraint is a value list of three forms.

We use two classes ValueList and ValueListPart to model value list and its composition. ValueListPart is an abstract class, the concrete subclasses are ValueLiteral, SemiboundedRange, and BoundedRange which response to three constraint forms.

ValueLiteral class has two attributes: datatype and value. The value attribute represents literal/lexical, and datatype attribute represents the datatype of that value.

SemiboundedRange class has three attributes: datatype, value, and isUpperBound. The usage and semantic of value and datatype attributes are same as above. If the value of isUpperBound is true, all values of object/role are equal or less than the value of value; else means equal or greater than this value.

BoundedRange class has five attributes: datatype, begin, end, includeBegin, and includeEnd. begin and end represents the top border and bottom border of value constraint; includeBegin and includeEnd indicate whether the value range border includes the value of begin or end.

D. Mandatory and Uniqueness Constraint

Figure 6 shows the class diagram of mandatory and uniqueness constraint. Mandatory and uniqueness constraints are applied to one or more object roles. We use two classes Mandatory and Uniqueness to model them.

![Diagram of Mandatory and Uniqueness Constraint](image)

**Figure 7.** Class diagram of mandatory and uniqueness constraint

XOR constraint is a mandatory plus exclusive constraint, a logical attribute isXOR can identify whether a mandatory is also a XOR constraint. Uniqueness has two attributes: isPrimary and isExternal; isPrimary represents whether the uniqueness constraint is a primary or not; isExternal represents whether the uniqueness constraint is an external or not.

E. Value-compare Constraints

Figure 8 shows the class diagram of value-compare constraint. The value-compare constraint restricts on two ordered object-roles called first role and second role. We use ValueCompare class and two associates with ObjectRole class to model value-compare constraint.

![Diagram of Value-compare Constraint](image)

**Figure 8.** Class diagram of value-compare constraint

There are five kinds of value compare operators: equal, greater than, greater than or equal, less than or equal, and less than. An enumeration class ValueCompareKind can model these compare operators. In ValueCompare class, type attribute represents selection a compare operator kind from ValueCompareKind. The semantic of a value-compare constraint on first role and second role is that values of these two roles must meet the following expression: (value of First) compare operator (value of Second).

F. Role-set Constraints: Subset, Equality, and Exclusion

Figure 9 shows the class diagram of role-set constraints. Because subset, equality, and exclusion constraints are all applied to two or more role-sets, they can be called as role-set constraints. Three classes Subset, SetEquality, and SetExclusion are response to three role-set constraints. A container class RoleSet is used to represent role-set, each role-set consists of one or more roles.

A role-set has at most one equality/exclusion constraint, and each equality/ exclusion constraint applied to at least two role-sets. A subset constraint only restricts on two ordered
role-sets, we use role name parent and child on these two associates to identify which is father role-set and which is child role-set.

![Figure 9. Class diagram of role-set constraints](image)

G. Frequency and Cardinality Constraints

Figure 10 shows the class diagram of frequency and cardinality constraints. Frequency constraint restrict times of one or more roles occurs in facts; and cardinality constraint restrict times of an object occurs in facts. We use two classes Frequency and Cardinality to model these two constraints, Frequency associates to ObjectRole (at least one), and Cardinality associate to ObjectType (exactly one).

![Figure 10. Class diagram of frequency and cardinality constraint](image)

These two constraint types have same restriction forms: exactly n (a positive integer), at least n, at most n, and between m and n (at least m and at most n). A frequency/cardinality constraint can only belong to one form of four. We define an enumeration class RangeKind to represents four restriction forms, the enumerate literal is: exactly-n, at least-n, at most-n, and range-m-n.

Frequency and Cardinality have same attributes, include two numerical attributes m and n and a RangeKind type attribute rangeType. The value of m is effective only when the value of rangeType is range-m-n.

H. Ring Constraints

Figure 11 shows the class diagram of ring constraint. A ring constraint only applies to two roles within a fact type which forms a cycle path between object(s) and these two roles. The kinds of ring constraint are followings: irreflexive, acyclic, intransitive, symmetric, asymmetric, antisymmetric, acyclic+intransitive, symmetric+irreflexive, symmetric+intransitive, asymmetric+intransitive, and purely reflexive.

We define a enumeration class RingKind to represent all kinds of ring constraint. RingConstraint class is used to model ring constraint; it has a RingKind type attribute ringType to represent which kind of ring constraint it is. RingConstraint associates to ObjectRole (exactly 2).

![Figure 11. Ring constraint](image)

VI. CONCLUSION

Based on informed researches and the proposed metamodel of ORM 2.0, this paper presented a MOF-compliant ORM-ODM for ORM 2.0, this ODM was validated by MagicDraw UML 16.5 and a responded XMI file was generated to interchange ORM-ODM. Based on ORM-ODM XMI file, An ORM-ML (XML schema for ORM metamodel) was generated by Enterprise Architect 8.0; this ORM-ML was validated in Altova XMLSpy 2010, and an example of ORM Schema was created based on it.

Our feature works are: improving ORM-ODM, creating algorithms to map ORM-ODM to CL-ODM, DL-ODM etc.

**REFERENCES**

5. OMG, "MOF 2.0/XMI Mapping, Version 2.1.1": Available online at: http://www.omg.org/spec/XMI/2.1/PDF.