Mapping ORM into the SHOIN/OWL Description Logic

-Towards a Methodological and Expressive Graphical Notation for Ontology Engineering

Mustafa Jarrar

Department of Computer Science, University of Cyprus
STARLab, Vrije Universiteit Brussels, Belgium**

Abstract. We map ORM into the $\mathcal{SHOIN}$/OWL, which is the most common description logic in ontology engineering. As $\mathcal{SHOIN}$/OWL is known to be a good compromise between expressiveness and computational complexity, this implies that the ORM constraints mapped in this paper are the constraints that are easier to implement and reason about. Our mappings are implemented as an extension to the DogmaModeler tool, which uses Racer as a background reasoning engine. Furthermore, the expressive, methodological, and graphical capabilities of ORM make it a good candidate for use as a graphical notation for ontology languages. In this way, industrial experts who are not IT savvy will still be able to build and view ontologies without needing to know the logic or reasoning foundations underpinning them.

Published As: *Mustafa Jarrar: Mapping ORM into the SHOIN/OWL Description Logic -Towards a Methodological and Expressive Graphical Notation for Ontology Engineering. OTM Workshops (ORM'07). LNCS 4805, Springer. 2007 http://www.jarrar.info/Publications/*

1 Motivation

In a previous work [15] we have mapped ORM into the $\mathcal{DLR}_{ifd}$[4], which is one of the most expressive description logics. Thanks to this expressivity, we have shown that all ORM constructs and constraints are decidable, except two rare cases[1]. $\mathcal{DLR}_{ifd}$ was developed indeed to allow the majority of the database primitives to be represented [3], including n-ary relations, identities, and functional dependencies. However, the problem is that not all $\mathcal{DLR}_{ifd}$'s constructs are implemented by current reasoning engines.

In this paper we map ORM into the $\mathcal{SHOIN}$ description logic, which is the logic underpinning OWL, the standard (W3C recommendation) Ontology Web Language. Compared with $\mathcal{DLR}_{ifd}$, $\mathcal{SHOIN}$ was developed as *a compromise between expressive power and decidability*[12]. This implies that the ORM mappings into $\mathcal{SHOIN}$ are easier to implement and exploit. $\mathcal{SHOIN}$/OWL is supported in almost all reasoning engines, and it is the most popular language in ontology engineering. However, the problem is that not all ORM constructs can be represented in $\mathcal{SHOIN}$/OWL. As we shall explain later, $\mathcal{SHOIN}$/OWL does not support n-ary relations, external uniqueness, among other things.

As a result, while the ORM mappings into $\mathcal{DLR}_{ifd}$ in [15] (which are 27 mapping rules out of the 29 ORM constructs) show which ORM constraints are decidable, the ORM mappings into $\mathcal{SHOIN}$/OWL in this paper (which are 22 mapping rules) show which ORM constraints are easier to implement and reason about, in case performance[2] is concerned.

** Soon the author will be affiliated only with the university of Cyprus.

[1] These cases are: a frequency spanning more than one role, and the acyclic ring constraint.

[2] Notice that performance is critical mostly for ORM query languages, e.g., ConQuer and RDIL.

Our motivation of mapping ORM into $\mathcal{SHOIN}$/OWL is not only to allow automated reasoning on ORM schemes [15] [20], but also to enable ORM to be used as graphical notation for ontology engineering. In many domains the ontology building process is difficult and time consuming. From practical experience, this is not only because these domains are not well understood or that a consensus cannot be made about them. In addition, it is usually difficult for domain experts to understand and use ontology languages. Current ontology languages (and *tools*) require an understanding of their underpinning logic. The limitation of these types of ontology languages is not that they lack expressiveness or logical foundations, but their suitability for being used by subject matter experts. The main requirements for an ontology language to be easily understood by domain experts are: (1)Its constructs should be close to the language that domain experts speak and the "logic" they use. For example, it is easier for a lawyer to say "it is mandatory for each Complaint Problem to be testified by at least one Evidence", than to say "the cardinality between "Complaint Problem" and "Evidence" is (1:0)". (2)The language should have a graphical notation to enable simple and conceptual modeling. By "graphical notation" here, we refer to not only *visualization* as some ontology tools offer, but also a *graphical language* that allow domain experts to construct an ontology using a notation for each concept, relation, and axiom. In other words, such a language should guide experts through its modeling constructs to "think" conceptually while building, modifying, or validating an ontology.

ORM has been used commercially for more than 30 years as a database modeling methodology, and has recently become popular not only for ontology engineering but also as a graphical notation in other areas such as the modeling of business rules, XML-Schemes design, data warehouses, requirements engineering, web forms, etc[3].

ORM has an expressive and stable graphical notation. It supports not only n-ary relations and reification, but also a fairly comprehensive treatment of many "practical" and "standard" business rules types. Furthermore, compared with, for example, EER or UML, ORM's graphical notation is more stable since it is attribute-free; in other words, object types and value types are both treated as concepts. This makes ORM immune to changes that cause attributes to be remodeled as object types or relationships.

Compared with other modeling notations, ORM diagrams can be automatically verbalized into pseudo natural language sentences. In other words, all rules in a given ORM diagram can be translated into fixed-syntax sentences [25]. For example, the subset constraint in section 4.8 is verbalized as: "*If a Person Drives a Car then this Person must be AuthorizedWith a DrivingLicense*". From a methodological viewpoint, this verbalization capability simplifies communication with non-IT domain experts and allows them to better understand, validate, or build ORM diagrams. It is worthwhile to note that ORM is the historical successor of NIAM (Natural Language Information Analysis Method), which was explicitly designed (in the early 70's) to play the role of a stepwise methodology, that is, to arrive at the "semantics"of a business application's data, based on natural language communication.

ORM's formal specification and semantics are well-defined (see e.g. [9] [10] [26] [6]). The most comprehensive formalization in first-order logic (FOL) was carried out by Halpin in [9] and [10]. Since reasoning on FOL is far too complex, namely undecidable[1], these formalizations do not enable automated reasoning on ORM diagrams. To enable such automated reasoning, we map all ORM semantics into $\mathcal{DLR}_{ifd}$ and $\mathcal{SHOIN}$/OWL Description Logics. Our mapping is based on the ORM syntax and semantics specified in [9] and [10].

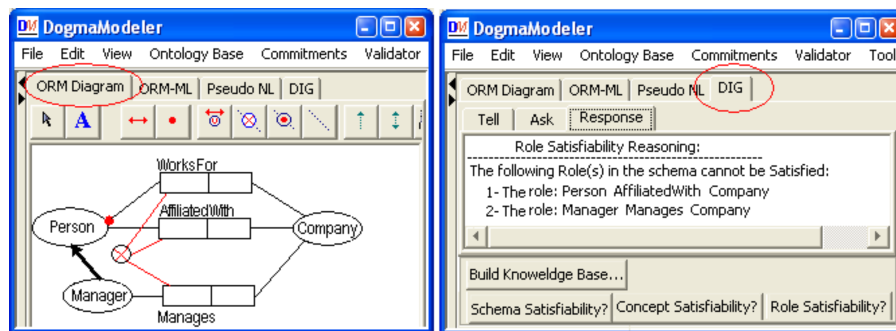
---

[3] Many commercial and academic tools that support ORM solutions are available, including the ORM solution within Microsoft's Visio for Enterprise Architects, VisioModeler, NORMA, CaseTalk, Infagon, and DogmaModeler. DogmaModeler and its support for ontology engineering will be presented later in this paper.

The remainder of the paper is organized as follows. Before presenting the technical details of our mappings, section 2 illustrates the implementation of these mappings as an extension to DogmaModeler. In section 3, we give a quick overview of related work. Section 4 maps ORM into $\mathcal{SHOIN}$/OWL. Finally, the conclusions are presented in section 5.

2 Implementation

Before presenting the technical details of mapping the ORM semantics into $\mathcal{SHOIN}$ /OWL, this section demonstrates the implementation of these mappings as an extension to the DogmaModeler [13]. DogmaModeler is an ontology modeling tool based on ORM. In DogmaModeler, ORM diagrams are mapped automatically into DIG, which is a DL interface, i.e., an XML serialization of DL statements that most reasoners (such as Racer, FaCT++, etc) support[4]. DogmaModeler is integrated with the Racer DL reasoning server, which acts as a background reasoning engine. In the screenshots below, the first window shows an ORM diagram, while the second DIG window shows the *Tell* and *Ask* functionalities. While the purpose of Tell is to map an ORM diagram into a knowledge base inside Racer, The purpose of Ask is to reason about this knowledge base. The results of the reasoning about the displayed ORM diagram indicate that the roles “AffiliatedWith” and “Manages” cannot be satisfied, because the mandatory and the exclusion constraints are conflicting each other. DogmaModeler currently implements three types of reasoning services: schema satisfiability, concept satisfiability, and role satisfiability[5]. Other types of reasoning services that are being implemented or are scheduled to be implemented include constraint implications, inference, and subsumption. Please refer to [19] for the technical details of mapping ORM into DIG.



3 Related work

Remark: In this paper, we focus only on the *logical* aspects of reusing ORM for ontology modeling. The conceptual aspects (i.e. ontology modeling versus data modeling) are discussed in [22] [13] [14] [18] [21], while a case study that uses the ORM notation

[4] Notice that DogmaModeler maps ORM into DIG directly, i.e., it does not map ORM into the OWL syntax at first place. This is because reasoning about an OWL ontology requires mapping it to DIG anyway. However, a functionality to export OWL syntax in DogmaModeler will be released the near future.

[5] These types of TBox reasoning in DgomaModeler are indeed very fast, we have evaluated this by reasoning on the CContology, a medium-size ontology about customer complaints[16] (200 concepts, 300 relations, 100 constraints). The reasoning took less than a second.

can be found in [17]. Similar to our work, there have been several efforts to reuse the graphical notation of UML and EER for ontology modeling. Some approaches, such as [23], considered this to be a visualization issue and did not consider the underpinning semantics. Others (e.g. [24]) are motivated only to detect consistency problems in conceptual diagrams. We have found the most decent work in formalizing UML in [2] and formalizing EER in [1]. These two formalization efforts have studied the FOL semantics of UML and EER and mapped it into $\mathcal{DLR}_{ifd}$. It is also worth noting that the ICOM tool was one of the first tools to enable automated reasoning with conceptual modeling. ICOM [7] supports ontology modeling using a graphical notation that is a mix of the UML and the EER notations. ICOM is fully integrated with the FaCT reasoning server, which acts as a background inference engine.

4 Mapping the ORM semantics into $\mathcal{SHOIN}$/OWL

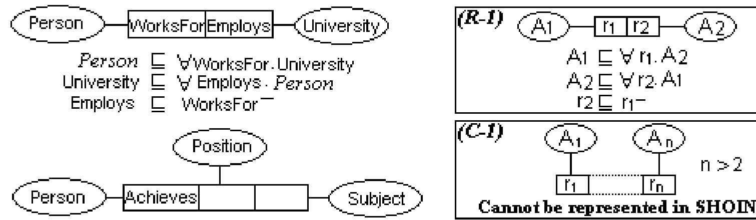
This section presents the ORM semantics in details. Every construct in ORM is discussed and mapped into $\mathcal{SHOIN}$ with a clear motivation for the mapping choices. In the following paragraphs we give a quick overview of $\mathcal{SHOIN}$.

Description logics are a family of knowledge representation formalisms. Description logics are decidable fragments of first-order logic, associated with a set of automatic reasoning procedures. The basic primitives of a description logic are the notion of a concept and the notion of a relationship. Complex concept and relationship expressions can be built from atomic concepts and relationships. For example, one can define $HumanMother$ as $Female \sqcap \exists HasChild.Person.$ The expressiveness of a description logic is characterized by the set of constructors it offers.

$\mathcal{SHOIN}$ is an expressive description logic [12]. It is the logic underpinning OWL, the standard (W3C recommendation) Ontology Web Language. $\mathcal{SHOIN}$ was developed as a compromise between expressive power and decidability. The $\mathcal{SHOIN}$ syntax is described as follows. If C and D are concepts and R is a binary relation (also called role), then ($C \sqcap D$), ($C \sqcup D$), ($\neg C$), ($\forall R.C$), and ($\exists R.C$) are also concepts. If R is simple (i.e., neither transitive nor has any transitive sub relations), then ($\leq nR$) and ($\geq nR$) are also concepts, where n is a non-negative integer. For C and D (possibly complex) concepts, $C \sqsubseteq D$ is called a general concept inclusion. $\mathcal{SHOIN}$ also allows hierarchies of relations ($R \sqsubseteq S$), transitivity (R_+), and inverse ($S \sqsubseteq R^-$). A recent extension [11] of $\mathcal{SHOIN}$ (called $\mathcal{SROIQ}$) enables representing more advanced constructs on relations, such as complex inclusions (e.g. $S \circ R \sqsubseteq R$), disjointness, negation ($\neg R$), and local reflexivity ($\exists R.Self$). Please refer to [12] for the semantics of $\mathcal{SHOIN}$, and [11] for more details on $\mathcal{SROIQ}$.

ORM supports n-ary relationships, where $n \geq 1$. Each argument of a relationship in ORM is called a *role*. For example, the binary relationship below has two roles, $WorksFor$ and $Employs$. The formalization of the general case of an ORM n-ary relationship[9] is: $\forall x_1...x_n(R(x_1...x_n) \rightarrow A_1(x_1) \wedge ... \wedge A_n(x_n))$. $\mathcal{SHOIN}$ supports only binary relationships. As shown in the example below, we represent a role in ORM as a relationship in $\mathcal{SHOIN}$; thus to represent a relation in ORM an additional axiom is required to state that both relations are inverse to each other. Rule *R-1* maps ORM binary relations into $\mathcal{SHOIN}$. *C-1* shows the general case of an ORM n-ary relations, which cannot be represented into $\mathcal{SHOIN}$; see [15] on how to represent this case using the $\mathcal{DLR}$ description logic.[6].

[6] There are 22 mapping rules (*R-1...R-22*) in this paper that maps ORM into $\mathcal{SHOIN}$. The cases that cannot be mapped into $\mathcal{SHOIN}$ but can be mapped into $\mathcal{DLR}$ are labeled as (*C-1...C-7*). There are 2 cases that lead to undecidability (thus cannot be mapped into any description logic) which are labeled as *Exception-1* and *Exception-2*.

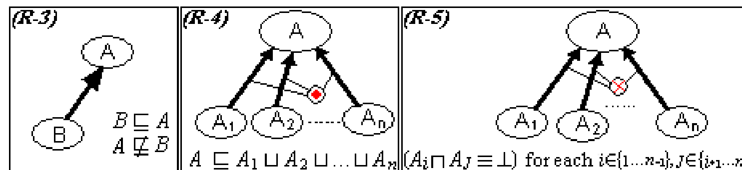


ORM allows unary roles, which is not allowed in $\mathcal{SHOIN}$. The example below means that a person may smoke; or in FOL [9]: $\forall x(Smokes(x) \rightarrow Person(x))$. The population of this fact is either true or false. To map ORM unary roles into $\mathcal{SHOIN}$, we introduce a new class called BOOLEAN, which takes one of two values: {TRUE, FALSE}. Each ORM unary fact is seen as a binary relationship in $\mathcal{SHOIN}$, where the second concept is BOOLEAN. Rule *R-2* presents the general case mapping of ORM unary fact types.

Remark: When mapping an ORM schema into a $\mathcal{SHOIN}$ knowledge base: Each role in ORM should have a unique label within its relation. In case a role label is null, an automatic label is assigned, such as r_1, r_2. In case of a relation having the same labels of its roles, e.g., $ColleagueOf/ColleagueOf$, new labels are assigned to these roles, such as: $ColleagueOf_1$, $ColleagueOf_2$.

4.1 Subtypes

Subtypes in ORM are proper subtypes. We say B is a proper subtype of A *iff* the population of B is always a subset of the population of A, and A $\neq$ B. This implies that the subtype relationship is acyclic; hence, loops are illegal in ORM. To map this in $\mathcal{SHOIN}$, we introduce an additional negation axiom for each subtype. For example, (Man Is-A Person) in ORM is mapped as: $(Man \sqsubseteq Person) \sqcap (Person \not\sqsubseteq Man)$. Rule *R-3* presents the general case mapping of ORM subtypes. Notice that "$\not\sqsubseteq$" is not part of the $\mathcal{SHOIN}$ syntax. However, it can be implemented by reasoning on the ABox[7] to make sure that the population of A and the population B are not equal.



4.2 Total Constraint The total constraint ($\odot$) between subtypes means: the population of the supertype is exactly the union of the population of these subtypes (see rule *R-4*).

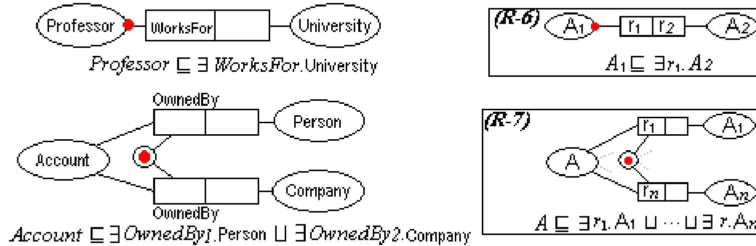
4.3 Exclusive Constraint The exclusive constraint ($\otimes$) between subtypes means the population of these subtypes is pairwise distinct, i.e. the intersection of the population of each pair of the subtypes must be empty (see rule *R-5*).

4.4 Mandatory Constraints

[7] Abox is a set of instantiation assertions, or in other words it is the set of instances (i.e. not the schema, which is called the TBox).

4.4.1 Role Mandatory. The role mandatory constraint is depicted as a dot on the line connecting a role with an object type. The example below indicates that, in every interpretation of this schema, each instance of the object-type Professor must work for at least one University. Rule *R-6* presents the general case mapping.

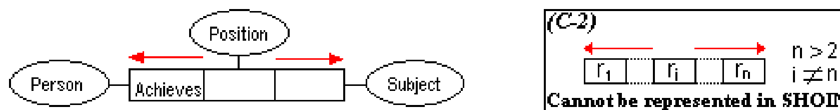
4.4.2 Disjunctive Mandatory. The disjunctive mandatory constraint is used to constrain a set of two or more roles connected to the same object type. It means that each instance of the object type's population must play at least one of the constrained roles. For example, the disjunctive mandatory in the example below means: each account must be owned by at least a person, a company, or both. Rule *R-7* presents the general case.



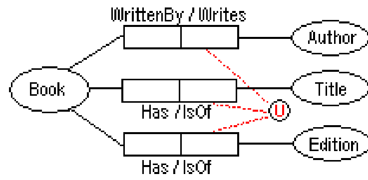
4.5 Uniqueness Constraints

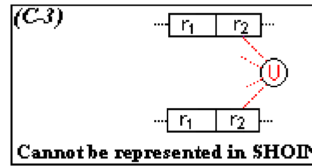
4.5.1 Role Uniqueness. Role uniqueness is represented by an arrow spanning a single role in a binary relationship. As shown in the example, the uniqueness states that, in every interpretation of this schema, each instance of a Professor must work for at most one University, i.e. each occurrence is unique. (See rule *R-8*).

4.5.2 Predicate Uniqueness. An arrow spanning more than a role in a relationship of arity n represents *predicate uniqueness*. As shown in the example below, the uniqueness constraint states that, in any population of this relationship, the person and subject pair must be unique together. The general case of this constraint *C-2* is formalized in FOL[9] as: $\forall x_1, .., x_i, .., x_n, y(R(x_1, .., x_i, .., x_n) \wedge R(x_1, .., y, x_{i+1}, .., x_n) \rightarrow x_i = y)$. This case cannot be represented in $\mathcal{SHOIN}$, but it can be represented using the notion of functional dependency in $\mathcal{DLR}_{idf}$ (See [15]).



4.5.3 External Uniqueness. External uniqueness constraints (denoted by "U") apply to roles from different relationships. The roles that participate in such a uniqueness constraint uniquely refer to an object type. As shown in the example below, the combination of (Author, Title, Edition) must be unique. In other words, different values of (Author, Title, Edition) refer to different Books. The formalization of the general case [9] of this constraint(see *C-3*) is: $\forall x_1, x_2, y_1..y_n(R_1(x_1, y_1) \wedge ... \wedge R_n(x_1, y_n) \wedge (R_1(x_2, y_1) \wedge ... \wedge R_n(x_2, y_n) \rightarrow x_1 = x_2)$. Also this case cannot be represented in $\mathcal{SHOIN}$, but it can be represented using the notion of identity in $\mathcal{DLR}_{idf}$ (See [15]).

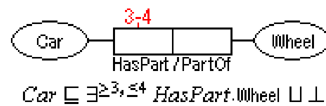


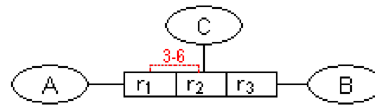


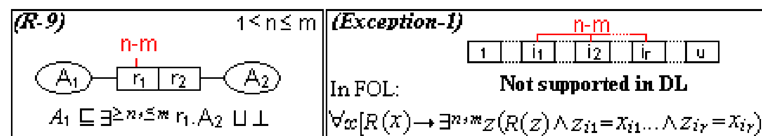
4.6 Frequency Constraints

4.6.1 Role Frequency Constraints. A frequency constraint $(min - max)$ on a role is used to specify the number of occurrences that this role can be played by its object-type. A frequency constraint on the ith role of an n-ary relation is formalized [9] as: $\forall x[x \in R.i \rightarrow \exists^{n,m} z(R(z) \wedge z_i = x)]$. For example, the frequency constraint in the example below indicates that *if* a car has wheels, then it must have at least 3 and at most 4 wheels. We map this constraint by conjugating $\perp$ to the $(min - max)$ cardinality, i.e. either there is no occurrence, or it must be within the $(min - max)$ range, which is the exact meaning in ORM. Rule *R-9* presents the general case mapping.

4.6.2 Multiple-role Frequency Constraints. A multiple-role frequency constraint spans more than one role (see the second example). This constrain means that, in the population of this relationship, A and C must occur together (i.e. as a tuple) at least 3 times and at most 6 times. Up to our knowledge, such a cardinality constraint cannot be formalized in description logic. However, this constraint is extremely rare in practice, [10] presents an example of this constraint and shows that it can be remodeled and achieved by a combination of uniqueness and single-role frequency constraints, which are indeed cheaper to compute and reason about. *Exception-1* presents the general case of a multiple-role frequency constraint and its formalization in first order logic [9].

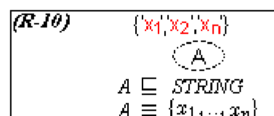


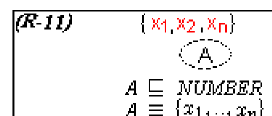




4.7 Value Constraints

The value constraint in ORM indicates the possible values (i.e. instances) for an object type. A value constraint on an object type A is denoted as a set of values $\{s_1, ..., s_n\}$ depicted near an object type, which indicate that $(\forall x[A(x) \equiv x \in \{s_1, ..., s_n\}])$ [9]. Value constraints can be declared only on lexical object types LOT, and values should be well-typed, i.e. its datatype should be either a string such as $\{'be', '39', 'it', '32'\}$ or a number such as $\{1, 2, 3\}$. Notice that quotes are used to distinguish string values from number values. If a LOT has no value constraint on it, then it is, by default, seen as a subtype of LEXICAL. If it has a value constraint, it must be a subtype of either the STRING or the NUMBER classes.

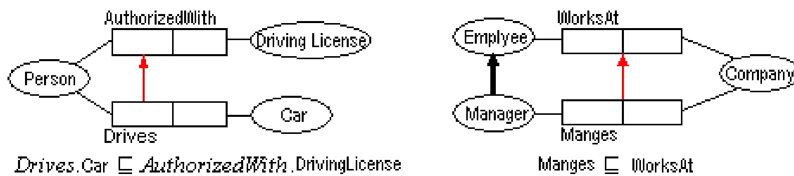




4.8 Subset Constraint

The subset constraint ($\rightarrow$) between two roles is used to restrict the population of these roles as one is a subset of the other. The first example below indicates that each person who drives a car must be authorized by a driving license: $\forall x(x \in R2.Drives \rightarrow x \in R1.AuthorizedWith)$ [9]. Rule *R-12* presents the general case mapping into $\mathcal{SHOIN}$. The second example indicates a subset between two relations, i.e., the set of tuples of the subsuming relation is a subset of the tuples of the subsumed relation (see *R-13*).

ORM also allows subset constraints between tuples of (not necessarily contiguous) roles as shown in case *C-3*, where each ith and j^{th} roles must have the same type. The population of the set of the jth roles is a subset of the population of the set of the ith roles. The FOL formalization of the general case of this constraint [9] is : $\forall x_1...x_k[\exists y(R_2(y) \wedge x_1 = y_{i_1} \wedge ... \wedge x_k = y_{i_k}) \rightarrow \exists z(R_1(z) \wedge x_1 = z_{j_1} \wedge ... \wedge x_k = z_{j_k})]$. This type of constraints cannot be represented in $\mathcal{SHOIN}$, not only because $\mathcal{SHOIN}$ does not support n-ary relations but also because $\mathcal{SHOIN}$ does not support projections on relations. See our mappings of this constraint into $\mathcal{DLR}_{Lite}$ [15].

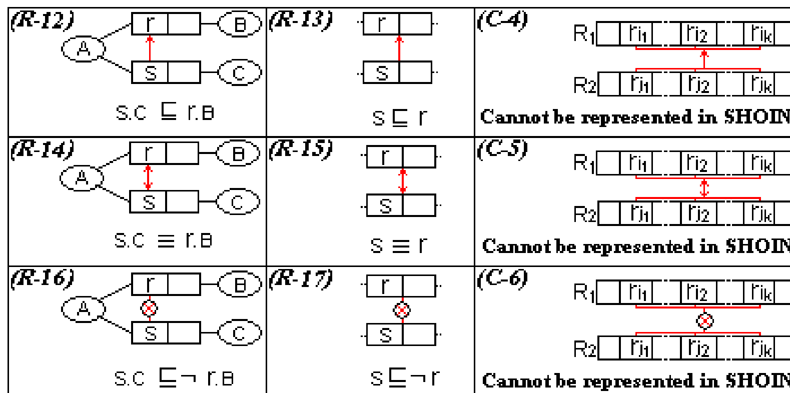


4.9 Equality Constraint

Similar to the subset constraint, the equality constraint ($\leftrightarrow$) between roles or relations are mapped as shown in rules *R-14* and *R-15* respectively.

4.10 Exclusion Constraint

Similar to the subset and quality constraints, the exclusion constraint ($\otimes$)between roles or relations are mapped as shown in rules *R-16* and *R-17*.



4.11 Ring Constraints

ORM allows ring constraints to be applied to a pair of roles (i.e. on binary relations) that are connected directly to the same object-type, or indirectly via supertypes. Six types of ring constraints are supported by ORM.

4.11.1 Symmetric (sym). This constraint states that if a relation holds in one direction, it should also hold on the other, such as "colleague of". R is symmetric over its population *iff* $\forall x, y[R(x, y) \longrightarrow R(y, x)]$. The example shown in rule *R-18* illustrates the

symmetric constraint and its general case mapping into $\mathcal{SHOIN}$.

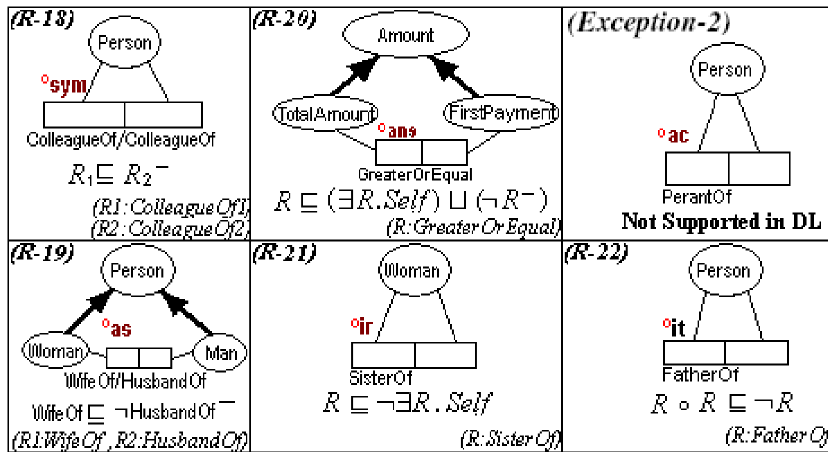
4.11.2 Asymmetric (as). Asymmetric is opposite of symmetric. If a relation holds in one direction, it cannot hold on the other; e.g., "wife of" and "parent of". R is asymmetric over its population *iff* $\forall xy, R(x, y) \longrightarrow \neg R(y, x)$ The example shown in rule *R-19* illustrates the asymmetric constraint and its general case mapping into $\mathcal{SHOIN}$.

4.11.3 Antisymmetric (ans). The antisymmetric constraint is also an opposite to the symmetric constraint, but not exactly the same as asymmetric; the difference is that all asymmetric relations must be irreflexive, which is not the case for antisymmetric. R is antisymmetric over its population *iff* $\forall xy, x \neq y \wedge R(x, y) \longrightarrow \neg R(y, x)$ (see the example in rule *R-20*). To map this constraint we use the concept $(\exists R.Self)$ that has been introduced recently to the $\mathcal{SROIQ}$ description logic [11]. The semantics of this concept simply is: $(\exists R.Self)^I = \{x \mid < x, x > \in R^I\}$.

4.11.4 Irreflexive (ac). This constraint states that an object cannot participate in a relation with himself. For example, a person cannot be the "parent of" himself. R is Irreflexive over its population *iff* $\forall x, \neg SisterOf(x, x)$. As discussed above, mapping this constraint into DL is also possible using the concept $\exists R.Self$. See rule *R-21*.

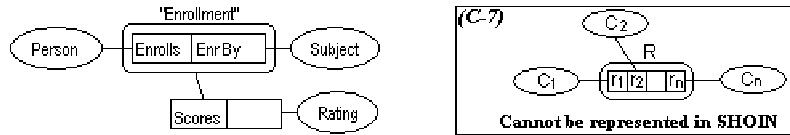
4.11.5 Acyclic (ac). The acyclic constraint is a special case of the irreflexive constraint; for example, a Person cannot be directly (or indirectly through a chain) ParentOf himself. R is acyclic over its population *iff* $\forall x[\neg Path(x, x)]$. In ORM, this constraint is preserved as a difficult constraint. "Because of their recursive nature, acyclic constraints maybe expensive or even impossible to enforce in some database systems."[10]. Indeed, even some highly expressive DLs support notions such as n-tuples and recursive fixed-point structures, from which one can build simple lists, trees, etc. However, to our knowledge, acyclicity with any depth on binary relations cannot be represented.

4.11.6 Intransitive (ac). A relation R is intransitive over its population *iff* $\forall x, y, z[R(x, y) \wedge R(y, z) \longrightarrow \neg R(x, z)]$. If Person X is FatherOf Person Y, and Y is FatherOf Z, then it cannot be that X is FatherOf Z. We map this constraint using the notion of *role-composition*. The composition of the two relations R and S (written as $R \circ S$) is a relation, such that: $R^I \circ S^I = \{(a, c) | \exists b.(a, b) \in R^I \wedge (b, c) \in S^I\}$. Hence, any composition with R itself $(R \circ R)$ should not imply R, see rule *R-22*.



4.12 Objectified Relations An objectified relation in ORM is a relation that is regarded as an object type, receives a new object type name, and is depicted as a rectangle around the relation. In the following example, each (Person, Subject) enrollment is treated as an object type that scores a rating. The general case of predicate objects (see *C-7*) in ORM is formalized in [9] as: $\forall x[A(x) \equiv \exists x_1, ..., x_n(R(x_1, ..., x_n) \wedge x = (x_1, ..., x_n))]$. In

addition to this axiom, it is assumed that there must be a uniqueness constraint spanning all roles of the objectified relation, although it is not explicitly stated in the diagram, see [10] for more details. Objectified relations cannot be represented in $\mathcal{SHOIN}$ as the additional uniqueness axiom cannot be represented in $\mathcal{SHOIN}$; See [15] on how to represent objectified relations in $\mathcal{DLR}$.



5 Conclusions and future work

We have mapped an extensive set of the ORM constructs into $\mathcal{SHOIN}$, which is the most poplar description logic and the logic underpinning the standard Ontology Web Language (OWL). Although $\mathcal{SHOIN}$ does not allow *all* ORM constructs to be represented, but as $\mathcal{SHOIN}$ is known to be a good a compromise between expressive power and decidability/computation, this implies that this set of ORM (i.e. the 22 mapping rules) is easy to implement and reason about. Indeed, ORM based query languages would inherit this compromise in their expressiveness and query processing.

The mappings presented in this paper would not only empower ORM with reasoning services, but they also empower $\mathcal{SHOIN}$/OWL itself with an expressive and methodological graphical notation. Using ORM as a graphical notation to model $\mathcal{SHOIN}$/OWL ontologies would simplify the ontology modeling process, specially because ORM - unlike other graphical notations- is an expressive and attribute-free language, and can be verbalized into natural language sentences. Indeed, we experienced this power of ORM in modeling the CContology (an e-business ontology of customer complaints), which we have developed with about 50 lawyers and domain experts (see [16]). These people were able to build this ontology with only 2 tutorials (each is 1 hour) about ORM, and without needing to know the underpinning logic behind this ontology.

In the future, we plan to study the reasoning complexity of *each* ORM constraint; extend the ORM graphical notation to include some $\mathcal{SHOIN}$ notions that are not supported in ORM, such as, the composition of relations, transitive closure, equivalentClass, datatypes, and intersection and union between relations. We also plan to implement additional types of reasoning services, specifically constraint implications and inferencing; and, develop a functionality in DogmaModeler to export and import OWL, experiments in [5] are valuable for this functionality.

From our experience, the 22 ORM constructs in this paper are the most popular in ontology engineering. However, we plan to run an experiment on the Swoogle[8] ontologies to know how much ORM is capable of representing these ontologies. We expect ORM to be powerful in domains such as Business, Finance, and Human Resources, where ontologies are taxonomies or knowledge schemes. ORM (and even $\mathcal{SHOIN}$) might not be powerful enough in domains such as Life Sciences, where axiomatizations are very fine-grained. See our report [8] on future OWL extensions.

Acknowledgment: I am indebted to Enrico Franconi for his valuable suggestions and contributions, and to Sergio Tessaris, Rob Shearer, and Terry Halpin for their valuable comments and feedback on some parts of this paper. I wish to thank Diego Calvanese, Maurizio Lenzerini, Robert Meersman, Alessandro Artale, Erik Proper, Marijke Keet, Ian Horrocks, and Stijn Heymans for their comments and suggestions during this research. This research is partially supported by the Knowledge Web project (FP6-507482) and the SEARCHiN project (FP6-042467, Marie Curie Actions).

[8] http://swoogle.umbc.edu/ (visited Septemebr 2007)

References

1. Franz Baader, Diego Calvanese, Deborah McGuinness, and Daniele Nardiand Peter Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.
2. Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on uml class diagrams. *Artificial Intelligence*, 168(1):70–118, 2005.
3. D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. *Proc. of the ACM SIGACT-SIGMOD-SIGART*, pages 149–158, 1998.
4. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Identification constraints and functional dependencies in description logics. In *the IJCAI'01*, pages 155–160, 2001.
5. Bach D., Meersman R., Spyns P., and Trog D. Mapping owl-dl into orm/ridl, (to appear). In *OTM 2007 Workshops, proceeding of ORM'07*. Springer Verlag, November 2007.
6. Olga de Troyer. A formalization of the binary object-role model based on logic. *Data and Knowledge Engineering*, 19:1–37, 1996.
7. Enrico Franconi and Gary Ng. The i.com tool for intelligent conceptual modelling. In *7th Int. WS on Knowledge Representation meets Databases*. Springer, 2000.
8. Stoilos G, Stamou G, Shearer S, Horrocks I, Pan J, and Jarrar M. Requirements for further language extensions, d2.5.4. Technical report, KnowledgeWeb-IST-2004-507482, 2006.
9. Terry Halpin. *A logical analysis of information systems: static aspects of the data-orientedperspective*. PhD thesis, University of Queensland, Brisbane, Australia, 1989.
10. Terry Halpin. *Information Modeling and Relational Databases*. Morgan-Kaufmann, 2001.
11. Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning*, 2006.
12. Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *Proceedings of the (LPAR'99)*, pages 161–180. Springer, 1999.
13. Mustafa Jarrar. *Towards Methodological Principles for Ontology Engineering*. PhD thesis, Vrije Universiteit Brussel, Brussels, Belgium, May 2005.
14. Mustafa Jarrar. Towards the notion of gloss, and the adoption of linguistic resources informal ontology engineering. In *Proceedings of the 15th international conference on World Wide Web (WWW2006)*, pages 497–503, Edinburgh, Scotland., May 2006. ACM Press.
15. Mustafa Jarrar. Towards automated reasoning on orm schemes. In *Proceedings of the 26th International Conference on Conceptual Modeling (ER 2007)*. Springer, 2007.
16. Mustafa Jarrar. *Towards Effectiveness and Transparency in e-Business Transactions, An Ontology for Customer Complaint Management*, chapter 8. Idea Group Inc., 2007.
17. Mustafa Jarrar. *Towards Effectiveness and Transparency in e-Business Transactions, An Ontology for Customer Complaint Management*. Idea Group Inc., 2007.
18. Mustafa Jarrar, Jan Demey, and Robert Meersman. On using conceptual data modeling for ontology engineering. *Journal on Data Semantics (Special issue on Best papers from the ER/ODBASE/COOPIS2002 Conferences.)*, 2800:185–207, October 2003.
19. Mustafa Jarrar and Mohammed Eldammagh. Reasoning on orm using racer. Technical report, Vrije Universiteit Brussel, Brussels, Belgium, August 2006.
20. Mustafa Jarrar and Stijn Heymans. Towards pattern-based reasoning for friendly ontology debugging. *International Journal on Artificial Intelligence Tools*, 17(4), August 2008.
21. Mustafa Jarrar and Robert Meersman. Formal ontology engineering in the dogma approach. In Robert Meersman and Zahir Tari, editors, *Proceedings of the International Conference on Ontologies, Databases andApplications of Semantics (ODBase 02).*, volume LNCS 2519, pages 1238–1254. Springer Verlag, 2002.
22. Mustafa Jarrar and Robert Meersman. *Ontology Engineering -The DOGMA Approach*, volume 1 of *Advances in Web Semantics*, chapter 3. Springer, 2008.
23. P., S. Cranefield, L. Hart, M. Dutra, K.Baclawski, M. Kokar, and J. Smith. Uml for ontology development. *Knowl. Eng. Rev.*, 17(1):61–64, 2002.
24. J. Simmonds and M.C. Bastarrica. A tool for automatic uml model consistency checking. *Proc of the IEEE/ACM on Automated software engineering*, pages 431–432, 2005.
25. Halpin T. and Curland M. Automated verbalization for orm 2. In *OTM'06 Workshops*.
26. Proper-H. van der Weide T ter Hofstede, A. Formal definition of a conceptual language for the description and manipulationof information models. *Info Sys*, 18(7):471–495, 1993.