Towards Automated Reasoning on ORM Schemes
-Mapping ORM into the $\mathcal{DLR}_{idf}$ description logic

Mustafa Jarrar

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

Abstract. The goal of this article is to formalize Object Role Modeling (ORM) using the $\mathcal{DLR}$ description logic. This would enable automated reasoning on the formal properties of ORM diagrams, such as detecting constraint contradictions and implications. In addition, the expressive, methodological, and graphical capabilities of ORM make it a good candidate for use as a graphical notation for most description logic languages. In this way, industrial experts who are not IT savvy will still be able to build and view axiomatized theories (such as ontologies, business rules, etc.) without needing to know the logic or reasoning foundations underpinning them. Our formalization in this paper is structured as 29 formalization rules, that map all ORM primitives and constraints into $\mathcal{DLR}$, and 2 exceptions of complex cases. To this end, we illustrate the implementation of our formalization as an extension to DogmaModeler, which automatically maps ORM into DIG and uses Racer as a background reasoning engine to reason about ORM diagrams.

Published As: *Mustafa Jarrar: Towards Automated Reasoning on ORM Schemes -Mapping ORM into the $\mathcal{DLR}_{idf}$ description logic. Proc. of the 26th Int. Conf. on Conceptual Modeling (ER 2007). Volume 4801, LNCS, Pages (181-197), Springer. 2007. http://www.jarrar.info/Publications/*

1 Motivation and Background

This article proposes to formalize ORM (Object Role Modeling [8]) using the $\mathcal{DLR}$ description logic. This would enable automated reasoning to be carried out on the formal properties of ORM diagrams, such as detecting constraint contradictions and implications. In addition, the expressive, methodological, and graphical power of ORM make it a good candidate for use as a graphical notation for most description logic languages. With this, non-IT trained industrial experts will be able to build axiomatized theories (such as ontologies, business rules, etc.) in a graphical manner, without having to know the underpinning logic or foundations.

ORM is a conceptual modeling method that allows the semantics of a universe of discourse to be modeled at a highly conceptual level and in a graphical manner. ORM has been used commercially for more than 30 years as a database modeling methodology, and has recently becoming popular not only for ontology engineering but also as a graphical notation in other areas such as the modeling of business rules, XML-Schemes, data warehouses, requirements engineering,web forms, etc[1].

** The author is currently moving from Brussels to Nicosia and soon will be affiliated only with the university of Cyprus.

[1] 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.

ORM has an expressive and stable graphical notation. It supports not only n-ary relations and reification, but as will be shown in this article it supports a fairly comprehensive treatment of many "practical" and "standard" business rules and constraint 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.

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. For example, the mandatory constraint in section 2.3 is verbalized as: "*Each Professor must WorksFor at least one University*". The subset constraint in section 2.8 is verbalized as: "*If a Person Drives a Car then this Person must be AuthorizedWith a DrivingLicense*". Additional explanation can be found in [21] and [26], which provide sophisticated and multilingual verbalization templates. 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.

Indeed, the graphical expressiveness and the methodological and verbalization capabilities of ORM makes it a good candidate for a graphical notation for modeling and representing ontologies and their underpinning logic.

ORM's formal specification and semantics are well-defined (see e.g. [7][27][28][5]). The most comprehensive formalization in first-order logic (FOL) was carried out by Halpin in [7]. Later on, some specific portions of this formalization were reexamined, such as subtypes[11], uniqueness[9], objectification[10], and ring constraints [8]. Since reasoning on first order logic is far complex, namely undecidable[1], the above formalizations do not enable automated reasoning on ORM diagrams, which comprises e.g. detection of constraint contradictions, implications, and inference.

In [19] and [20] , we presented a reasoning approach based on heuristics (called pattern-based reasoning) for detecting the common constraint contradictions in ORM. This approach was designed to be user friendly and easy to apply in interactive modeling. It indicates not only constraint contradictions, but also a clear explanation about the detected contradictions, the causes, and suggestions on how to resolve these contradictions. Although this reasoning approach is easy to apply specially by non-IT domain experts, in comparison with DL-based reasoning, but it cannot be complete. In other words, there is no guarantee that by passing the predefined patterns, that the ORM schema is satisfiable. Please refer to [20] for more details on this approach and for a comparison (and a synergy) between the pattern-based and the DL-based reasoning mechanisms.

Enable automated and complete reasoning can only be done in description logic. This papers maps all ORM primitives and constraints into the $\mathcal{DLR}_{ifd}$ Description Logic, which is an expressive and decidable fragment of first-order logic. Our mapping is based on the ORM syntax and semantics specified in [7] and [8].

The remainder of the paper is organized as follows. In the following section, we give a quick overview about the $\mathcal{DLR}$ description logic. Section 2 presents the complete formalization of ORM using $\mathcal{DLR}$. In section 3, we illustrate the implementation of this formalization as an extension to DogmaModeler and present some related work. Finally, the conclusions and directions for future work are presented in section 4.

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 [23] [14] [15] [17] [22], while a case study that uses the ORM notation can be found in [16].

The $\mathcal{DLR}$ Description Logic

Description logics are a family of logics concerned with knowledge representation. A description logic is a decidable fragment of first-order logic, associated with a set of automatic reasoning procedures. The basic constructs for a description logic are the notion of a concept and the notion of a relationship. Complex concept and relationship expressions can be constructed from atomic concepts and relationships with suitable constructs between them. The expressiveness of a description logic is characterized by the constructs it offers. The simplest description logic is called $\mathcal{FL}^-$[1], which offers only the intersection of concepts, value restrictions, and a simple form of existential quantification. In other words, a *TBox* in $\mathcal{FL}^-$ is built as a set of inclusion assertions of the following forms: $C, D \rightarrow A \mid C \sqcap D \mid \forall R.C \mid \exists R$.

In this paper, we use the $\mathcal{DLR}_{ifd}$ description logic[4], which is an extension to $\mathcal{DLR}$. $\mathcal{DLR}_{ifd}$ is an expressive description logic, and allows the majority of the primitives and constraints used in data modeling to be represented [1], including n-ary relations, identification, and functional dependencies. The basic constructs of $\mathcal{DLR}$ are concepts and n-ary relations ($n \geq 2$). Let A denote an atomic concept, and P an atomic n-ary relation. Arbitrary concepts, denoted by C in $\mathcal{DLR}$ and arbitrary relations denoted by R, can be built according to the following syntax respectively: $C ::= \top_1 \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid (\leq k[i]R)$, and $R ::= \top_n \mid P \mid (i/n : C) \mid \neg R \mid R_1 \sqcap R_2$, where n denotes the arity of the relations P, R, R_1 and R_2, i denotes a component of a relationship, and k denotes a non-negative integer. Relations in $\mathcal{DLR}$ are *well-typed*, which means that only relations of the same arity n can be used in expressions like $R_1 \sqcap R_2$ and $i \leq n$ whenever i denotes a component of a relation of arity n. The following are abbreviations: $\bot$ for $\neg\top_1$; $C_1 \sqcup C_2$ for $\neg(\neg C_1 \sqcap \neg C_2)$; $C_1 \Rightarrow C_2$ for $\neg C_1 \sqcup C_2$; $(\leq k[i]R)$ for $\neg(\leq k-1\ [i]R)$; $\exists[i]R$ for $(\geq 1[i]R)$; $\forall[i]R$ for $\neg\exists[i]\neg R$; and $(i : C)$ for $(i/n : C)$ if n is clear from the context.

The semantics of $\mathcal{DLR}$ is specified as follows. An *interpretation* I is constituted by an interpretation domain $\triangle^I$, and an interpretation function $.^I$ that assigns to each concept C a subset C^I of $\triangle^I$ and to each R of arity n a subset R^I of $(\triangle^I)^n$. $t[i]$ denotes the i-th component of tuple t.

$$\begin{array}{rlrl}
\top_n^I & \subseteq (\triangle^I)^n & \top_1^I & = \triangle^I \\
P^I & \subseteq \top_n^I & A^I & \subseteq \triangle^I \\
(i/n : C)^I & = \{t \in \top_n^I | t[i] \in C^I\} & (\neg C)^I & = \triangle^I \backslash C^I \\
(\neg R)^I & = \top_n^I \backslash R^I & (C_1 \sqcap C_2)^I & = C_1^I \cap C_2^I \\
(R_1 \sqcap R_2)^I & = R_1^I \cap R_2^I & (\leq k[i]R)^I & = \{a \in \triangle^I \mid \sharp\{t \in R^I | t[i] = a\} \leq k\}
\end{array}$$

A $\mathcal{DLR}$ *TBox* is constituted by a finite set of inclusion assertions, where each assertion has the form: $C_1 \sqsubseteq C_2$ or $R_1 \sqsubseteq R_2$, with R_1 and R_2 of the same arity. Beside these inclusion assertions in $\mathcal{DLR}$, $\mathcal{DLR}_{ifd}$ allows identification id and functional dependencies fd assertions to be expressed, which have the following form: (**id** $C\ \ [r_1]R_1, ..., [r_n]R_n$) and (**fd** $R\ r_1, ..., r_h \rightarrow r_j$). Furthermore, another useful extension that has been recently included in $\mathcal{DLR}$-*Lite* [3] which we shall use in this paper, is inclusion between projections of relations, which has the following form: $R_2[r_{j_1}, ..., r_{j_k}] \sqsubseteq R_1[r_{i_1}, ..., r_{i_k}]$. Inclusion, identification id and functional dependencies fd shall be explained later in this paper.

2 The formalization of ORM using $\mathcal{DLR}_{ifd}$

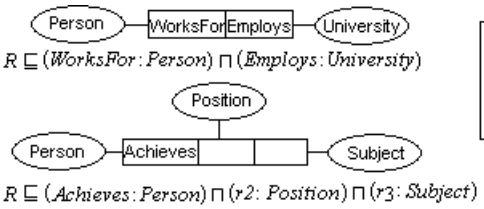
2.1 Object-Types

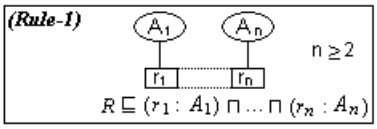
ORM allows a domain to be modelled by using object-types that play certain roles. There are two kinds of object-types in ORM: Non-Lexical Object-Types (NOLOT) and Lexical Object-Types (LOT). Both object-types are depicted as ellipses in ORM's notation. a LOT is depicted as a dotted-line ellipse and a NOLOT is a solid-line ellipse[2]. We represent both NOLOTs and LOTs as classes in $\mathcal{DLR}$. To distinguish between NOLOT and LOT in a $\mathcal{DLR}$ knowledge base, we introduce four classes: LEXICAL, STRING, NUMBER, and BOOLEAN. The class LEXICAL is considered to be a super-type of the other three classes, while the other three classes are considered to be disjoint. Unless specified, each LOT is mapped by default into the class STRING. We shall return to this issue later in the paper.

2.2 Roles and relationships

ORM supports n-ary relationships, where $n \geq 1$. Each argument of a relationship in ORM is called a *role*. The examples below show binary and ternary relationships. For example, the binary relationship has two roles, $WorksFor$ and $Employs$. The formalization of the general case of an ORM n-ary relationship[7] is: $\forall x_1 ... x_n (R(x_1 ... x_n) \rightarrow A_1(x_1) \wedge ... \wedge A_n(x_n))$. $\mathcal{DLR}$ supports n-ary relationships, where $n \geq 2$. Each argument of a relationship in $\mathcal{DLR}$ is called a *component* [1]. As shown in the examples below, we represent a relationship in ORM as a relationship in $\mathcal{DLR}$; thus, a role in ORM is seen as a component of a relationship in $\mathcal{DLR}$.

For people who are familiar with ORM, the formalization of ORM roles and relationships shown in the examples seems to be trivial. However, people who are familiar with description logics may not find it intuitive. This is because, unlike ORM, the components of relationships in description logics are typically not used and do not have linguistic labels. For example, one expects to see the binary relationship in the example below represented in description logic as, $Person \sqsubseteq \forall WorksFor.University$, and $University \sqsubseteq \forall Employs.Person$. In this case, both $WorksFor$ and $Employs$ are two different relationships. This formalization requires an additional axiom to state that both relations are inverse to each other: $WorksFor \sqsubseteq Employs^{-}$. ORM schemes formalized in this way are not only lengthy, but also become more complex when relationships other than binary are introduced. As will be shown later, our method of formalizing ORM roles and relationships will make the formalization of the ORM constraints intuitive and more elegant. Rule-1 formalizes ORM n-ary relations, where $n \geq 2$.



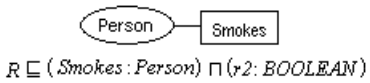


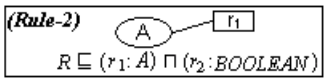
Remark: When mapping an ORM schema into a $\mathcal{DLR}$ knowledge base: Each role in the ORM schema should have a unique label within its relationship. In case a role label is null, an automatic label is assigned, such as r_1, r_2, etc. In case of a relationship having the same labels of its roles, such as $ColleagueOf/ColleagueOf$, new labels

[2] Although they are not exactly similar, the notions of LOT and NOLOT in ORM can be, for the sake of simplicity, compared to the concepts of 'Attribute' and 'Class' in UML.

are assigned to these roles, such as: $ColleagueOf - r_1$, $ColleagueOf - r_2$. Usually, ORM relationships do not have labels; thus, a unique label is automatically assigned, such as: R_1, R_2, etc.

ORM unary roles. Unlike $\mathcal{DLR}$, ORM allows the representation of unary relations. The relationship in the example below means that a person may smoke. The population of this role is either true or false. In first-order logic, this fact can be formalized [7] as: $\forall x(Smokes(x) \rightarrow Person(x))$. To formalize ORM unary roles in $\mathcal{DLR}$,we introduce a new class called BOOLEAN, which can take one of two values: either TRUE or FALSE. Each ORM unary fact is seen as a binary relationship in $\mathcal{DLR}$, where the second concept is BOOLEAN. Rule-2 presents the general case formalization of ORM unary fact types.



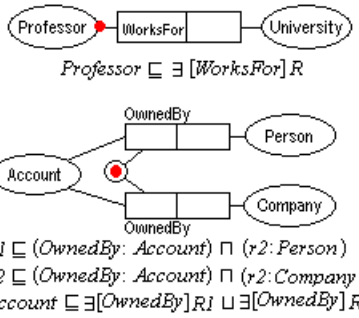


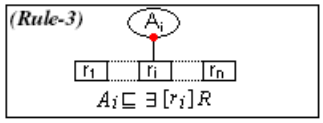
2.3 Mandatory Constraints

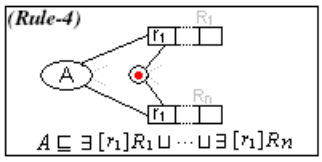
There are two kinds of mandatory constraints in ORM: roles and disjunctive.

Role Mandatory. The role mandatory constraint in ORM 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-3 presents the general case formalization of the role mandatory constraint.

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 that each account must be owned by at least a person, a company, or both. Rule-4 presents the general case formalization of a disjunctive mandatory constraint.





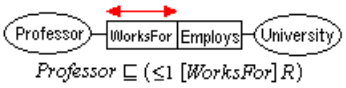


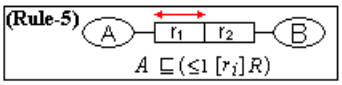
2.4 Uniqueness Constraints

We distinguish between three types of uniqueness constraints in ORM: role uniqueness, predicate uniqueness, and external uniqueness.

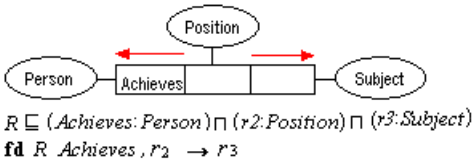
Role Uniqueness. Role uniqueness is represented by an arrow spanning a single role in a binary relationship. As shown in the example below, the uniqueness constraint states that, in every interpretation of this schema, each instance of a Professor must work for

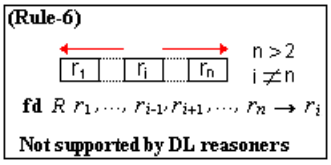
at most one University, i.e. each occurrence is unique. Rule-5 presents the general case formalization of the role uniqueness constraint.





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 is formalized in FOL[7] as: $\forall x_1, .., x_i, .., x_n, y(R(x_1, .., x_i, .., x_n) \wedge R(x_1, .., y, x_{i+1}, .., x_n) \rightarrow x_i = y)$. We formalize this uniqueness constraint using the notion of *functional dependency* **fd** in $\mathcal{DLR}_{ifd}$ [4], which has the form: (**fd** $R\ r_1, ..., r_h \rightarrow r_j$) ; where R is a relation, and $r_1, ..., r_h, r_j$ denote roles in R. The notion of functional dependency requires two tuples of a relationship that agree on the constrained components $r_1, ..., r_h$ to also agree on the un-constrained component r_j. The set of the constrained roles (on the the left-side of the **fd** assertion) uniquely determines the un-constrained role (which is on the the right side of the assertion).





Notice that our formalization excludes the following cases:

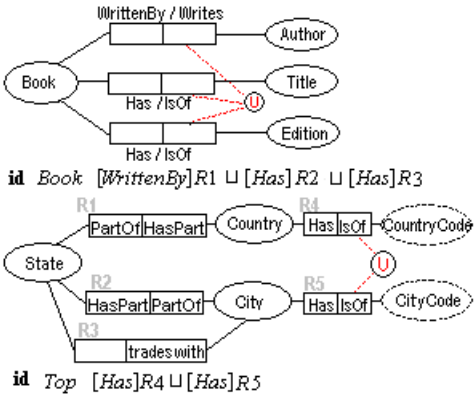
- Role uniqueness in a binary relationship: Although it is theoretically possible to use the above formalization in case of a binary relationship, we keep the formalization of this case separate (see rule-5) for implementation reasons. This is because: 1) rule-5 is supported in most description logic reasoners while rule-6 is not implemented in any reasoner yet, and 2) reasoning on functional dependencies cannot be performed on TBox only. In other words, as functional dependencies in $\mathcal{DLR}_{ifd}$ are seen as extra assertions (i.e. outside the TBox), the reasoning process to check whether the **fd** assertions are violated is reduced to ABox satisfiability. If there is no ABox, one cannot reason over the **fd** assertions.
- A single role uniqueness in an $n-$ary relationship where $(n > 2)$, since it is always a non-elementary fact type. This case is considered an illegal constraint in ORM (see [8], chapter 4), with [4] proving that it leads to undecidability in reasoning. Therefore, this case is ruled out in our formalization.

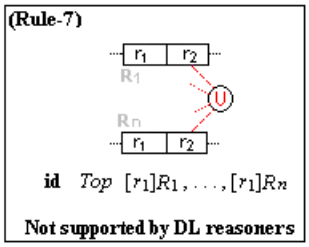
External Uniqueness. External uniqueness constraints (denoted by "U") apply to roles from different relationships. The roles that participate in such a uniqueness constraint uniquely refer to an object type. As shown in the example below, the combination of (Author, Title, Edition) must be unique. In other words, different values of (Author, Title, Edition) refer to different Books. Formalizing this constraint in description logic is possible using the notion of identity **id** in $\mathcal{DLR}_{ifd}$ [4]. In case the external uniqueness is defined on binary relationships and the common concept to be constrained is directly connected to these relations, the formalization is direct. In other cases, the formalization becomes more complex. We shall try to simplify and explain this complexity in the following.

The notion of identity **id** in $\mathcal{DLR}_{ifd}$ has the form: (**id** C $[r_1]R_1, ..., [r_n]R_n$), where C is a concept, each R_i is a relation, and each r_i is a role in R_i that is connected to C. The identity **id** in $\mathcal{DLR}_{ifd}$ states that two instances of the concept C cannot agree on the participation in $R_1, ..., R_n$ via their roles $r_1, ..., r_n$, respectively. See [4] for more details on this. In ORM, the intuition of external uniqueness is that the combination of $r_1, ..., r_n$ in $R_1, ..., R_n$ respectively must be unique. The formalization of the general case [7] of this constraint (see the figure in rule-7) is: $\forall x_1, x_2, y_1..y_n(R_1(x_1, y_1) \wedge ... \wedge R_n(x_1, y_n) \wedge (R_1(x_2, y_1) \wedge ... \wedge R_n(x_2, y_n) \rightarrow x_1 = x_2)$.

This allows one to define uniqueness on roles that are not directly connected to a *common concept*. For example, although the external uniqueness in the second example below means that the combination of {CountryCode, CityCode} must be unique, it does not tell us that the combination is unique for which concept. In other words, the notion of "common concept" is not explicitly regarded, neither in the ORM graphical notation nor in its underlying semantics [7] [9] [27]. To interpret the external uniqueness (i.e. the semantics) in this example, a join path should be performed on $R4 - R1$ and $R5 - R2$. In other words, although the notion of "common concept" does not exist in ORM, it is assumed that there must be a join path between the constrained roles. If this path cannot be constructed, then the external uniqueness is considered illegal [9], i.e. an error in the ORM schema. The construction of such join paths becomes more complex (even for human eyes) in large schemes or when objectified (i.e. reified) predicates are involved. [28] shows many cases of complex external uniqueness.

We formalize the general case of external uniqueness using the notion of **id** in $\mathcal{DLR}_{ifd}$, but we use the concept Top as the common concept C (see rule-7). As shown in the examples, the formalization (using Top) means that *any* two individuals must agree on their participation in roles: [WrittenBy]R1, [Has]R2 and [Has]R3. Although the use of the Top concept yields a simple and elegant formalization, intensive ABox reasoning may be required. In practice, we recommend using the Uniquest algorithm [28]. This algorithm is designed to compute the shortest join path connecting the constrained roles for an external uniqueness constraint, no matter what its level of complexity is. The result is a *derived relation*, which represents the shortest join path. This derived relation can then be used instead of the concept Top in rule-7.



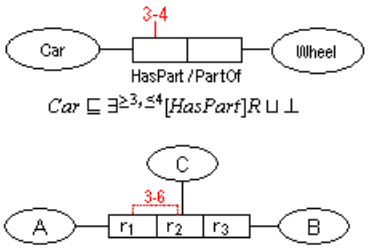


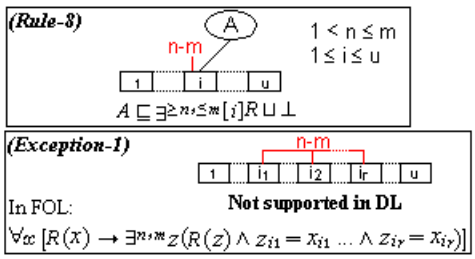
2.5 Frequency Constraints

In the following we formalize the Frequency Constraints. We distinguish between frequency constraints that span 1) a single role, which we call "role frequency" constraints, and 2) multiple roles, which we call "multiple-role frequency" constraints.

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 [7] 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 formalize 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-8 presents the general case mapping rule of a role frequency constraint.

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, [8] 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 [7].





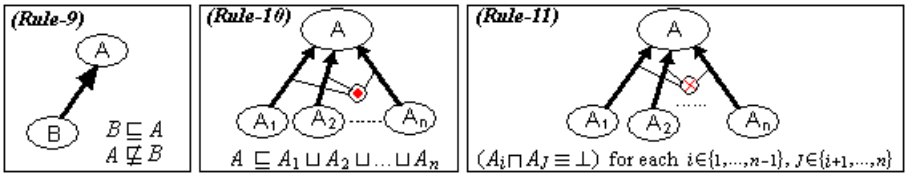
2.6 Subtypes

Subtypes in ORM are proper subtypes. For example, we say that B is a proper subtype of A if and only if 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 formalize this relationship in $\mathcal{DLR}$, we introduce an additional negation axiom for each subtype relation. For example, (Man Is-A Person) in ORM is formalized as: $(Man \sqsubseteq Person) \sqcap (Person \not\sqsubseteq Man)$. Rule-9 presents the general case formalization of ORM subtypes. Notice that "$\not\sqsubseteq$" is not part of the $\mathcal{DLR}$ syntax. However, it can be implemented by reasoning on the ABox to make sure that the population of A and the population B are not equal.

Remark: Subtypes in ORM should be *well-defined*, which means that users should introduce some rules explicitly to define a subtype. Such definitions are not part of the graphical notation and are typically written in the FORMAL language [7]. The idea of the ORM FORMAL language is similar to the idea the OCL language for UML. For example: if one states that (Man Is-A Person), then a textual rule on Man is defined e.g. *"who has Gender='Male"'*. Since such rules are not part of the graphical notation, we do not include them in our formalization. We assume that textual rules that are not part of the ORM graphical notation are written in $\mathcal{DLR}$ directly.

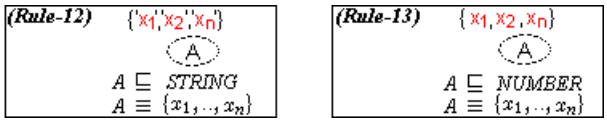
Total Constraint The total constraint ($\odot$) between subtypes means that the population of the supertype is exactly the union of the population of these subtypes (see rule-10).

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-11).



2.7 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 $(\forall x[A(x) \equiv x \in \{s_1, ..., s_n\}])$ [7]. 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. As discussed earlier, 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.

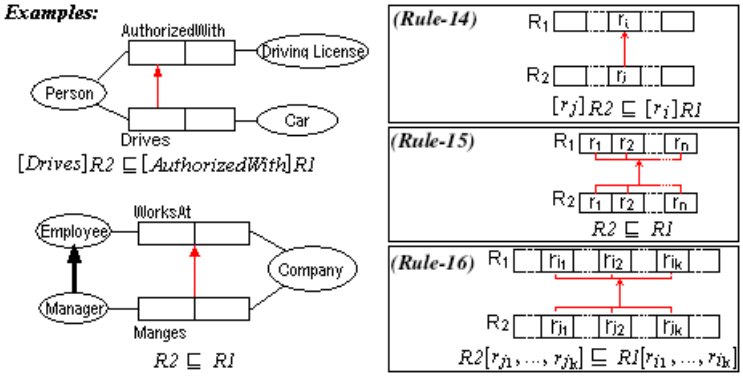


Outlook: We plan to extend our formalization of the ORM value constraint to include other data types, such as real, integer, and boolean, which are not discussed in this paper.

2.8 Subset Constraint

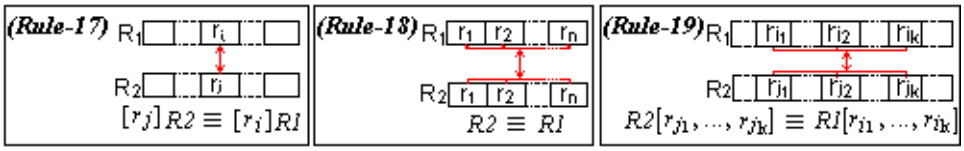
The subset constraint ($\rightarrow$) between roles (or sequences of roles) is used to restrict the population of these role(s), since one is a subset of the other. See the examples below. The first example indicates that each person who drives a car must be authorized by a driving license: $\forall x(x \in R2.Drives \rightarrow x \in R1.AuthorizedWith)$ [7]. If an instance plays the subsuming role, then this instance must also play the subsumed role. Rule-14 formalizes a subset constraint between two roles. A subset constraint that is declared between all roles in a relationship and all roles in another relationship implies that the set of tuples of the subsuming relation is a subset of the tuples of the subsumed relation. See the second example below. Rule-15 formalizes of a subset constraint between two relations. ORM also allows subset constraints between tuples of (not necessarily contiguous) roles as shown in rule-16, where each i^{th} and j^{th} roles must have the same type. The population of the set of the j^{th} roles is a subset of the population of the set of the i^{th} roles. The FOL formalization of the general case of this constraint [7] is : $\forall x_1...x_k[\exists y(R_2(y) \wedge x_1 = y_{i_1} \wedge ... \wedge x_k = y_{i_k}) \rightarrow \exists z(R_1(z) \wedge x_1 = z_{j_1} \wedge ... \wedge x_k = z_{j_k})]$.

To formalize this constraint in description logic, we use the recent extension to *DLR-Lite* [3] that allows inclusion assertions between *projections* of relations of the forms: $R_2[r_{j_1}, ..., r_{j_k}] \sqsubseteq R_1[r_{i_1}, ..., r_{i_k}]$, where R_1 is an n-ary relation, $r_{i_1}, ..., r_{i_k} \in \{r_1, ..., r_n\}$, and $r_{i_p} \neq r_{i_q}$ if $r_p \neq r_q$; R_2 is an m-ary relation, $r_{j_1}, ..., r_{j_k} \in \{r_1, ..., r_m\}$, and $r_{j_p} \neq r_{j_q}$ if $r_p \neq r_q$. Using this extension, any ORM set-comparison constraint formalized hereafter between two sets of (not contiguous) roles becomes direct. Rule-16 shows the subset general case.



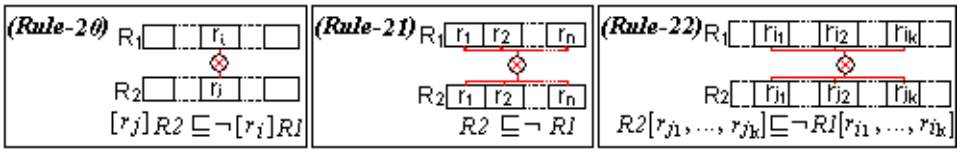
2.9 Equality Constraint

Similar to the subset constraint, the equality constraint ($\leftrightarrow$) between roles, relations, or sequences of roles is formalized in the following rules.



2.10 Exclusion Constraint

Similar to the subset and quality constraints, the exclusion constraint ($\otimes$) between roles, relations, or sequences of roles is formalized in the following rules.



2.11 Ring Constraint

In the following we formalize the 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: symmetric (sym), asymmetric (as), antisymmetric (ans), acyclic (ac), irreflexive (ir), and intransitive (it).

Symmetric Ring Constraint (sym). The symmetric constraint states that if a relation holds in one direction, it should also hold on the other direction, such as "colleague of" and "partner of". R is symmetric over its population *iff* $\forall x, y[R(x, y) \longrightarrow R(y, x)]$. The example shown in rule-23 illustrates the symmetric constraint and its general case formalization in $\mathcal{DLR}$.

Asymmetric Ring Constraint (as). The asymmetric constraint is the opposite of the symmetric constraint. If a relation holds in one direction, it cannot hold on the other; an example would be "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-24 illustrates the asymmetric

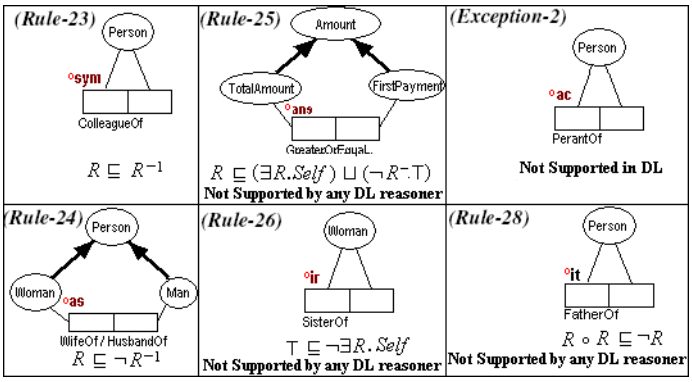
constraint and its general case formalization in $\mathcal{DLR}$.

Antisymmetric Ring Constraint (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-25). To formalize this constraint (and some other constraints below) in description logic, we use the concept $(\exists R.Self)$ that has been introduced recently to the $\mathcal{SROIQ}$ description logic and $\mathcal{RIQ}$ [12]. The semantics of this concept simply is: $(\exists R.Self)^I = \{x \mid< x, x >\in R^I\}$. Notice that this concept is not *yet* included in the $\mathcal{DLR}$ description logic that we use in this paper. However, as [12] shows, this concept can be added without causing difficulties in reasoning. Rule-25 illustrates the antisymmetric constraint and its general case formalization.

Irreflexive Ring Constraint (ac). The irreflexive constraint on a relation states that an object cannot participate in this relation with himself. For example, a person cannot be the "parent of" or "sister of" himself. R is Irreflexive over its population *iff* $\forall x, \neg SisterOf(x,x)$. As discussed above, formalizing this constraint in description logic is also possible using the concept $\exists R.Self$. Rule-26 illustrates the irreflexive constraint and its general case formalization in description logic.

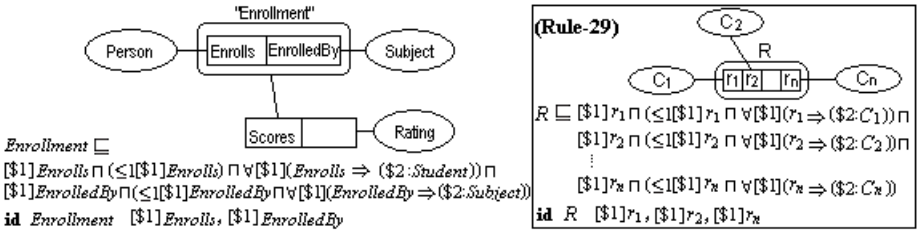
Acyclic Ring Constraint (ac). The acyclic constraint is a special case of the irreflexive constraint; for example, a Person cannot be directly (or indirectly through a chain) ParentOf himself. R is acyclic over its population *iff* $\forall x[\neg Path(x,x)]$. In ORM, this constraint is preserved as a difficult constraint. "Because of their recursive nature, acyclic constraints maybe expensive or even impossible to enforce in some database systems."[8]. Indeed, even some highly expressive description logics 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.

Intransitive Ring Constraint (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 formalize this constraint using the notion of *role-composition* in description logic. 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-28.



2.12 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. To help explain predicate objects in ORM, we use a familiar example (see figure 26 [8]). In this example, each (Person, Subject) enrollment is treated as an object that scores a rating. Predicate objects in ORM are also called objectified relationship types or nested fact types. The general case of predicate objects in ORM is formalized in [7] 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. This is to indicate that e.g. each person may enroll in many subjects, and the same subject may be enrolled by many persons; see [8] or the recent [10] for more details.

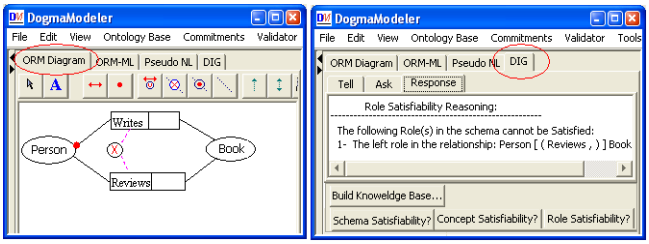
Predicate objects in ORM can be formalized using the notion of *reification* in $\mathcal{DLR}_{ifd}$. Reifying an n-ary relationship into a $\mathcal{DLR}_{ifd}$ concept is done by representing this concept with n binary relations, with one relationship for each role[2]. To understand this reification, one can imagine the "Enrollment" example by remodeled into two binary relations, one for the role "Enrolls" and one for the role "EnrolledBy". The new concept "Enrollment" is defined in the example below. In this definition: ($[\$1]Enrolls$ and $[\$1]EnrolledBy$) specify that the concept "Enrollment" must have all roles "Enrolls" and "EnrolledBy" of the relationship, ($\leq 1[\$1]Enrolls$ and $\leq 1[\$1]EnrolledBy$) specify that each of these roles is single-valued, and ($\forall[\$1](Enrolls \Rightarrow \$2 : Student)$ and $\forall[\$1]((EnrolledBy \Rightarrow \$2 : Subject)$) specify the object type each role belong to. The last identity **id** assertion is to specify a uniqueness constraint spanning all roles (i.e. "Enrolls" and "EnrolledBy"). Rule-29 presents the general case formalization of the objectified predicates in $\mathcal{DLR}_{ifd}$.



3 Implementation and Related Work

In this section, we illustrate the implementation of the formalization presented in this paper. The formalization is implemented as an extension to the DogmaModeler [14]. DogmaModeler is an ontology modeling tool based on ORM. In DogmaModeler, ORM diagrams are mapped automatically into DIG, which is a description logic interface (XML-based language) that most reasoners (such as Racer, FaCT++, etc) support. DogmaModeler is integrated with the Racer description logic reasoning server which acts as a background reasoning engine. See a screen shot of DogmaModeler below. The first window shows an ORM diagram, while the second window shows the reasoning results on this digram. The results indicate that the role "Person Reviews Book" cannot be satisfied. DogmaModeler currently implements three types of reasoning services: schema satisfiability, concept satisfiability, and role satisfiability. The other types of reasoning services that are being implemented or are scheduled to be implemented include constraint implications, inference, and subsumption. Please refer to [18] for the technical details of DogmaModeler's mapping into DIG.

The main problem we faced during the implementation is that several ORM constraints cannot be mapped into DIG; that is, these constraints were not yet supported by any description logic reasoner. Each formalization rule that could not be implemented is marked by *"Not supported by any DL reasoner yet"* in the previous section.



One may notice that, in the first place, we did not map ORM into OWL, the standard web ontology language. The reason is that OWL is based on a description logic called $\mathcal{SHOIN}$ [13], rather than the $\mathcal{DLR}_{ifd}$ that we use in this paper. Compared with $\mathcal{DLR}_{ifd}$, $\mathcal{SHOIN}$ does not support n-ary relations, identification, functional dependencies, and projection of relations, among other things. This implies that several ORM constraints cannot be formalized in $\mathcal{SHOIN}$, and thus cannot be supported in OWL. These constraints are: predicate uniqueness, external uniqueness, set-comparison constraints (subset, equality, and exclusion) between single roles and between not contiguous roles, objectification, as well as n-ary relationships.

Notice that without these constraints, mapping ORM into OWL becomes direct, based on our formalization. In other words, formalizing ORM using $\mathcal{SHOIN}$/OWL can be seen as a subset of the formalization presented in this paper. All formalization rules can hold for $\mathcal{SHOIN}$/OWL except {rules-6,7,14,16,17,19,20,22, and 29}. The syntax of some rules need to be modified such as Rule-1: $A1 \sqsubseteq \forall R.A2$, Rule-2: $A \sqsubseteq \forall R.BOOLEAN$, Rule-4: $A \sqsubseteq \exists R_1.C_1 \sqcup .... \sqcup \exists R_n.C_n$, etc. Actually, what DogmaModeler currently maps into DIG is what can be mapped into OWL. A DogmaModeler functionality to export OWL in this way (i.e. as a subset of ORM) will be released in the near future.

3.2 Related work

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 [24], considered this to be a visualization issue and did not consider the underpinning semantics. Others (e.g. [25]) are motivated only to detect consistency problems in conceptual diagrams. We have found the most decent work in formalizing UML in [2], and in [1] for EER. These two formalization efforts have studied the FOL semantics of UML and EER and mapped it into the $\mathcal{DLR}_{ifd}$ description logic, which we use in this paper. It is also worth noting that the ICOM tool was one of the first tools to enable automated reasoning with conceptual modeling. ICOM [6] 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 description logic reasoning server, which acts as a background inference engine.

4 Conclusions and future work

In this paper, we have formalized ORM using the $\mathcal{DLR}_{ifd}$ description logic. Our formalization is structured into 29 formalization rules which map all ORM primitives and constraints, except for two complex cases (see exception 1 and 2). We have shown

which formalization rules can be implemented by current description logic reasoning engines, and which can be mapped into $\mathcal{SHOIN}$/OWL. We have illustrated the implementation of our formalization as an extension to the DogmaModeler. Hence, we have explained how ORM can be used as as a graphical notation for ontology modeling with the reasoning being carried out by a background reasoning engine.

Various issues remain to be addressed. These include extending our formalization to cover more datatypes besides the String, Number, and Boolean types; implementing additional types of reasoning services, specifically constraint implications and inferencing; developing a functionality in DogmaModeler to export OWL; studying the computational complexity of ORM constraints; and last but least, is to extend the ORM graphical notation to include some description of logical notions, such as the composition of relations and intersection and union between relations.

Acknowledgment: This research was initiated during my visit to Enrico Franconi at the Free University of Bozen-Bolzano, which was funded by the Knowledge Web project (FP6-507482). I am indebted to Enrico for his very valuable suggestions, contributions, and encouragement. I am also indebted to Sergio Tessaris, Terry Halpin, and Rob Shearer for their valuable comments and feedback on the final version of this paper. I wish to thank Diego Calvanese, Maurizio Lenzerini, Stijn Heymans, Robert Meersman, Ian Horrocks, Alessandro Artale, Erik Proper, Marijke Keet, and Jeff Pan for their comments and suggestions during this research. This research is partially funded by the SEARCHiN project (FP6-042467, Marie Curie Actions).

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. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, MaurizioLenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. In Patrick Doherty, John Mylopoulos, and Christopher Welty, editors, *Proceedings of the 10th International Conference on Principles of KnowledgeRepresentation and Reasoning (KR2006)*, pages 178–218, Menlo Park, California, 2006. AAAI Press.
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. Olga de Troyer. A formalization of the binary object-role model based on logic. *Data and Knowledge Engineering*, 19:1–37, 1996.
6. Enrico Franconi and Gary Ng. The i.com tool for intelligent conceptual modelling. In *7th Int. WS on Knowledge Representation meets Databases*. Springer, 2000.
7. Terry Halpin. *A logical analysis of information systems: static aspects of the data-orientedperspective*. PhD thesis, University of Queensland, Brisbane, Australia, 1989.
8. Terry Halpin. *Information Modeling and Relational Databases*. Morgan-Kaufmann, 2001.
9. Terry Halpin. Join constraints. In Terry Halpin, Keng Siau, and John Krogstie, editors, *Proceedings of the 7th International IFIP WG8.1 Workshop on Evaluation ofModeling Methods in Systems Analysis and Design (EMMSAD'02)*, June 2002.
10. Terry Halpin. Objectification. In Terry Halpin, Keng Siau, and John Krogstie, editors, *Proceddings of the 10th International Workshop on Exploring Modeling Methodsin Systems Analysis and Design (EMMSAD05) at CAiSE 2005*, 2005.
11. Terry Halpin and Erik Proper. Subtyping and polymorphism in object-role modelling. *Data and Knowledge Engineering*, 15(3):251–281, 1995.
12. 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.
13. Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *Proceedings of the (LPAR'99)*, pages 161–180. Springer, 1999.

14. Mustafa Jarrar. *Towards Methodological Principles for Ontology Engineering*. PhD thesis, Vrije Universiteit Brussel, Brussels, Belgium, May 2005.
15. 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.
16. Mustafa Jarrar. *Towards Effectiveness and Transparency in e-Business Transactions, An Ontology for Customer Complaint Management*. Idea Group Inc., 2007.
17. 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.
18. Mustafa Jarrar and Mohammed Eldammagh. Reasoning on orm using racer. Technical report, Vrije Universiteit Brussel, Brussels, Belgium, August 2006.
19. Mustafa Jarrar and Stijn Heymans. Unsatisfiability reasoning in orm conceptual schemes. In Illarramendi A. and Srivastava D., editors, *Proceeeding of International Conference on Semantics of a Networked World.*, volume LNCS. Springer, March 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, Maria Keet, and Paolo Dongilli. Multilingual verbalization of orm conceptual models and axiomatized ontologies. Technical report, Vrije Universiteit Brussel, Brussels, Belgium, February 2006.
22. 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.
23. Mustafa Jarrar and Robert Meersman. *Ontology Engineering -The DOGMA Approach*, volume 1 of *Advances in Web Semantics*, chapter 3. Springer, 2008.
24. 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.
25. 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.
26. Halpin T. and Curland M. Automated verbalization for orm 2. In *OTM'06 Workshops*.
27. P. van Bommel, A. H. M. ter Hofstede, and T. P. van der Weide. Semantics and verification of object-role models. *Information Systems*, 16(5):471–495, 1991.
28. Th. P. van der Weide, A. H.M. ter Hofstede, and P. van Bommel. Uniquest: determining the semantics of complex uniqueness constraints. *Comput. J.*, 35(2):148–156, 1992.