BIRZEIT UNIVERSITY

Faculty of Information Technology

Master of Computing

On Using a Graphical Notation in Ontology Engineering

Prepared By
Rami Mohammad Awad Hodrob

Supervised By
Dr. Mustafa Jarrar

M.Sc. Master

BIRZEIT

2012

On Using a Graphical Notation in Ontology Engineering

استخدام الرسومات الدلالية في هندسة الانطولوجيا

Prepared By
Rami Mohammad Awad Hodrob

Supervised By
Dr. Mustafa Jarrar

This thesis was submitted in partial fulfillment of the requirements for the Master's Degree in Computing From the Faculty of Graduate Studies at Birzeit University, Palestine

BIRZEIT

Febreuary, 2012

BIRZEIT UNIVERSITY

Faculty of Graduate Studies

Master of Computing

Thesis Approval

On Using a Graphical Notation in Ontology Engineering

Prepared By: Rami Mohammad Awad Hodrob

Supervised By: Dr. Mustafa Jarrar

This Master Thesis was successfully defended in 11/02/2012

The names and signatures of the examining committee members as follows:

1. Head of committee: Dr. Mustafa Jarrar   Signature ....

2. Internal examiner: Dr. Wasel Ghanem   Signature ....

3. External examiner: Dr. Mahmoud Saheb   Signature ....

Birzeit-Palestine

February, 2012

Dedication

To my parents, to my family: Shatah, Mira, Mohammad, and Yahya Hodrob.
They taught me to always keep looking.

Rami Mohammad Hodrob

Acknowledgements

I am grateful to Dr. Mustafa Jarrar for his support and contribution to this research, particularly for the contributions he provided in the mapping of ORM into OWL2 and its underpinning description logic, in addition to the proposed extension of the existing ORM notation. I am also indebted to Anton Deik for providing a practical use case and contributing to the mapping/formalization process as well as the ORM extension in addition to revising major parts of the text of this thesis.

Also, I extend my thanks to the students of the Master program in Computing for their help and support especially in their active participation in the evaluation workshops conducted to evaluate the proposed ORM extension.

I am thankful to my parents, the spirit of my father Mohammad Hodrob, my mother and my wife Shatah for their immeasurable support throughout my life. Without their love, patience and support, I would have never come this far.

Table of Contents

List of Tables

List of Figures

List of Appendices

List of Abbreviations

- ABox : Assertion Box
- ALC : Attribute Concept Description Language with Complements
- DIG : Description Logic Implementation Group
- DLs : Description Logics
- EER : Enhanced Entity Relationship
- IDE : Integrated Development Enhancement
- IRI : Internationalized Resource Identifier
- LOT : Lexical Object Type
- NGO : Non-Governmental Organization
- NIAM : Natural Language Information Analysis Method
- nRQL : new RacerPro Query Language
- ODM : Ontology Definition Metamodel
- OKBC : Open Knowledge Base Connectivity protocol
- OMG : Object Management Group
- ORM : Object Role Modeling
- ORM-ML : ORM Markup Language
- OWL : Web Ontology Language
- RBox : Role Box
- RDF : Resource Description Framework
- SPARQL : Simple Protocol and RDF Query Language
- TBox : Terminological Box
- UML : Unified Modeling Language
- UNA : Unique Name Assumption
- UoD : Universe of Discourse
- URI : Uniform Resource Identifier
- W3C : World Wide Web Consortium
- XML : Extensible Markup Language

Publications

1. Rami Hodrob, Mustafa Jarrar: Mapping ORM into OWL 2. In proceedings of the *International Conference on Intelligent Semantic Web – Applications and Services*. Pages 68-73. ACM ISBN 9781450304757. June 2010.

2. Mustafa Jarrar, Rami Hodrob, Anton Diek: An ORM-based Graphical Notation for Authoring OWL 2 Ontologies. *Data & Knowledge Engineering Journal. Elsevier*. (Submitted, December 2011).

Abstract

The enormous need for well engineered ontologies is growing rapidly, as the need for ontologies is increasing in many application areas such as data integration, the semantic web, knowledge engineering, enhanced information retrieval, etc. Due to the central role ontologies are playing, the World Wide Web Consortium (W3C) developed the Web Ontology Language (OWL) as a language to author ontologies. However, OWL, like many other similar ontology languages, does not provide a practical and methodological means for ontology engineering. In addition, one is required to understand the logical foundation underpinning OWL, which is very difficult for domain experts. For an ontology language to be easily understood by domain experts it must be close to the natural language they speak and the 'logic' they use. Also, it should have a graphical notation to enable simple and conceptual modeling. The expressive, methodological, and graphical capabilities of Object-Role Modeling (ORM) make it a good candidate for use in ontology engineering. The modeling approach (ORM) selected here is one of the richest graphical modelling approaches and the knowledge and practice of it is easy to be acquired at a short period of time. The second version of OWL (OWL 2) is a recommended web ontology language from W3C which contains the majority of the constructs to be used for building any needed ontology. Many reasoners such as RacerPro 2.0, Pellet, Hermit, Fact++ and others support reasoning ontologies represented in OWL 2 which is created using the description logic SROIQ (characterized by expressivity and decidability). In this research, we (i) map the most commonly used ORM constructs to OWL 2 using SROIQ Description Logic and, on the other hand, we (ii) extend the ORM notation to cover all OWL 2 constructs not currently covered by ORM. By doing so, we combine the strengths of both ORM and the W3C-recommended Web Ontology Language (OWL). This creates a framework that allows one to engineer OWL ontologies graphically using ORM.

المستخلص

نمت الحاجة بصورة كبيرة مؤخرا لـ هندسة الانطولوجيا (علم ما هو موجود) في مجالات تطبيقية متعددة، ومن أهمها : تكامل البيانات، الشبكة الدلالية، هندسة المعرفة، وتعزيز استرجاع المعلومات. ونظرا للدور المهم الذي تلعبه الانطولوجيا، قام اتحاد W3C باستحداث لغة انطولوجيا الويب (OWL) كلغة تستخدم في هندسة الانطولوجيا. يعد الإصدار الثاني للغة الانطولوجيا OWL 2 من أغنى لغات بناء الانطولوجيا لاحتوائها على الغالبية العظمى من التراكيب التي تمكّن المستخدم من بناء الانطولوجيا المطلوبة، بالإضافة إلى احتوائها على العديد من الميزات الجديدة، إلا أن لغة الانطولوجيا OWL 2 مثل لغات الانطولوجيا الأخرى لا توفر وسيلة عملية أو منهجية لهندسة الانطولوجيا، حيث يحتاج المستخدم لهذه اللغة في بناء الانطولوجيا، إلى فهم البنية الأساسية والمنطق الخاص بها وهو أمر ليس بالسهل. ولكي يتم فهم لغة الانطولوجيا من قبل مستخدمي هندسة الانطولوجيا يجب أن تكون هذه اللغة قريبة من اللغة الطبيعية التي يتحدثون بها أ أو "المنطق" الذي يستخدمونه وأيضا بحاجة لبيئة رسومية تتيح القيام ببناء الانطولوجيا بطريقة بسيطة. تعتبر القدرات التعبيرية والمنهجية، والإمكانات الرسومية لأداة النمذجة ORM من الأسباب التي رشحتها بشكل أساسي ليتم استخدامها في هندسة الانطولوجيا. كما أن أداة ORM التي اخترناها في هذه الأطروحة كطريقة عملية ومنهجية لبناء الانطولوجيا بلغة OWL2 تعتبر من أغنى الطرق ذي الرسومات الدلالية التي يتم فهمها وممارستها بسلاسة . قمنا في هذه الأطروحةبما يلي : (أ) تمثيل تراكيب ORM بلغة OWL 2 وذلك عن طريق استخدام لغة المنطق SROIQ (ب) تطوير ORM لتمثيل كامل لغة OWL 2، وبذلك جمعنا بين نقاط القوة الخاصة بـ ORM ولغة OWL2 وهذا يخلق الإطار الذي يمكّن المستخدم من هندسة الانطولوجيا بلغة OWL 2 باستخدام الرسومات الدلالية الخاصة بـ ORM المطورة.

Chapter One

Introduction

1.1 Introduction and Motivation

Ontology engineering is one of the major challenges that brought the attention of the research community in the last decade. This is especially due to the growing need for well-engineered ontologies [G98] for many application areas such as data integration, the semantic web, knowledge engineering, enhanced information retrieval and others [G98, BGME07, CW05, ZWL08, F03, F04, HS10, JMY99, VHS10, ZLW08]. Due to the central role ontologies play in realizing the vision of the semantic web [DFV03, FH11, LH07], the World Wide Web Consortium (W3C) developed the Web Ontology Language (OWL) among its semantic web technology stack, as a language to author ontologies for the semantic web. OWL is based on Description Logic. In particular, the older version of OWL is based on SHOIN [MLL06] description Logic while the newest version (OWL 2 [HKP+09]) is based on SROIQ. OWL is fully supported by many description logic reasoning tools such as Racer [HO01], Pellet [PP04], Hermit[1], Fact++ [TH06].

However, like other ontology languages, using OWL to engineer ontologies is a difficult task as it does not provide a practical and methodological means for ontology engineering. In addition, one is required to understand the logical foundation underpinning OWL, which is very difficult for domain experts. In fact, the limitations of OWL and other similar languages are not that they lack expressiveness or logical foundations, but their suitability for being used by subject matter experts. For an ontology language to be easily understood by domain

---

[1] http://web.comlab.ox.ac.uk/people/boris.motik/HermiT/

experts, it should at least meet the following two requirements [J05, J07b]: (i) it must be close to the natural language the experts speak and the 'logic' they use. (ii) The language should have a graphical notation to enable simple and conceptual modeling. What we mean by 'graphical notation' here is not merely *visualization* but a *graphical language* that allows for ontology construction using notations for concepts, relations, and axioms. In other words, such language should guide domain experts to think conceptually while building, modifying, and validating an ontology [GH08, COL08].

Object-Role Modeling (ORM) is a conceptual modeling approach that has been in use since the early 1970s in database modeling, and has recently become popular in ontology engineering [JDM03]. Its expressive, methodological and graphical capabilities make it indeed one of the best candidates for building ontologies. Specifically, what distinguish ORM as a conceptual modeling approach are its simplicity, intuitiveness, stability, and verbalization capabilities, among many others. ORM simplifies the modeling process by using natural language, intuitive diagrams, and examples, and by examining information in terms of simple elementary facts and expressing them in terms of objects and roles [H89, H01].

Compared to other graphical modeling notations such as EER or UML, ORM is a stable modeling notation and more expressive [JDM03, GH08]. This is due to the fact that ORM makes no use of attributes (i.e., attribute-free). All facts are represented in terms of concepts (object-types or value-types) playing roles [H04a]. This makes ORM not impacted by changes that cause attributes to be remodeled as object types or relationships. Also, one of ORM's strongest features is its verbalization capabilities; ORM diagrams can be automatically verbalized into pseudo natural language [H04a] (See the example in section 3.1). The verbalization capability of ORM simplifies the communication with subject matter experts and allows them to better understand, validate, and build ORM diagrams. It is also important to note that ORM's verbalization techniques have been adopted in the business rules community and have become an OMG standard.

For ORM to be used as an Ontology Engineering methodology, the underlying semantics of the notation must be formally specified (using logic). Mapping of ORM using First Order Logic was done comprehensively in [J07a]. In addition, because FOL is not decidable (does

not enable automatic reasoning), ORM was formalized in [J07a, J07b] using less complex logic languages that allows for automatic reasoning, namely, DLR Description logic [J07a] and SHOIN/OWL description logic [HJ10, J07b]. This extensive work on the formalization of ORM has indeed played an important role in laying the foundation for using ORM in ontology engineering.

In order to fully establish ORM as a practical and methodological means for ontology engineering, we propose to combine the strengths of both ORM and the W3C-recommended Web Ontology Language (OWL 2) [HKP+09]. In short, we propose to map all ORM constructs to OWL 2 using SROIQ Description Logic and, on the other hand, extend the ORM notation to cover all OWL2 constructs not currently covered by ORM. By doing so, we exploit the advantages of both ORM as an intuitive graphical approach for conceptual modeling and OWL2 as a standard W3C-recommended language [BKMP09] for authoring ontologies. This creates a framework that allows one to author OWL 2 ontologies graphically using ORM.

In the extension part (extending ORM to completely represent OWL 2), each construct of OWL 2 is checked if it is mapped by ORM, if it is not a proposed ORM graphical notation is chosen to represent that of OWL 2 according to a surveyed evaluation process. The constructs of OWL 2 that are not covered by ORM and are used to extend ORM are Equivalent Classes, Disjoint Classes, Intersection of Class Expressions, Class Complement, Class Assertions, Individual Equality, Individual Inequality, positive object/data property assertion and negative object/data property assertion. In addition to these proposed notations, some of the constructs of OWL 2 are represented as non-notational expressions.

It is important to note here that the main purpose of this thesis is to develop an expressive and methodological graphical notation for OWL 2, which allows people to author OWL 2 ontologies graphically. This is presented in this thesis in two parts. In the first part, we investigate all ORM constructs by mapping/formalizing them into OWL 2 and its underpinning SROIQ Description Logic. Here ORM is used and not ORM 2 where ORM is the same as ORM 2 except that ORM 2 has modified shapes for graphical notations to occupy less space in modeling. Some new notations of ORM 2 are still not mature. ORM notations are less complicated than that of ORM 2, where here we concentrate in graphical notations that are

more users friendly. In the second part, we investigate OWL 2 constructs that do not have equivalent graphical notations in ORM and develop ORM-inspired graphical notations for them. By doing so, we have developed an ORM-based graphical notation that expresses OWL 2 completely. Because of this, all of our work in this paper is based on the semantics of OWL 2 not on ORM's semantics as in [J07a, J07b]. That is, the semantics of some ORM constructs were altered to adapt them to the semantics of OWL 2.

1.2 Thesis Statement and Objectives

The aim of this research is to map between Object Role Modeling (ORM) and OWL 2 Web Ontology Language to enable one to use ORM as a graphical notation in ontology engineering. Acquiring the knowledge to build an ontology graphically using ORM can be accomplished with minimal effort and time. However, ORM as a graphical modeling approach is not supported by currently available reasoners. OWL 2 is the recommended language for authoring ontologies. However, using the OWL 2 syntax is rather complicated. So an innovative way to solve this problem is proposed by using ORM as interface for OWL2 for authoring ontologies. We map between ORM and OWL 2 and extend ORM for complete representation of OWL 2. The result of mapping enables one to build ontology graphically and check the correctness of the built ontology using an appropriate reasoner that supports OWL2.

The objectives of the thesis are as follows:

- Mapping from ORM into SROIQ Description Logic/OWL 2.

- Extending the notations of ORM to completely represent OWL 2.

- Evaluating the correctness of mapping (ORM into OWL2), using appropriate reasoning services such as instance checking.

- Evaluating the proposed ORM extension using a survey.

- Extending DogmaModeler (modeling tool) to hold the implementation of mapping between ORM and OWL 2.

1.3 Contributions

The original contributions of this thesis can be summarized as follows:

1. Contributes to the mapping of ORM into SROIQ/OWL 2. This mapping was done for each of the twenty nine constructs that forms the ORM notations. This mapping was carried out jointly and in a close cooperation with Anton Deik and Dr Mustafa Jarrar, and based on previous research published in [J07a] and [J07b].

2. Contributes to extending the ORM notation by introducing new graphical notations to cover OWL 2 constructs not currently covered by ORM. The extended ORM notation covers all constructs of OWL 2. This extension was carried out jointly and in a close cooperation with Anton Deik and Dr Mustafa Jarrar.

3. Evaluating the correctness of our mapping, where each mapped ORM construct was loaded as a complete OWL/XML file into RacerPro 2.0. Different reasoning methods like consistency, coherency and special instance checking were used to prove the correctness of our mapping.

4. Evaluating the new ORM extension by means of a survey conducted with more than 31 ORM experts and practitioners.

5. Implementation: We extended DogmaModeler (an ORM-based modeling tool [JM08]) to implement (a) our mapping (from ORM into OWL 2) and (b) the new ORM extension. Also, we have integrated the Hermit reasoning tool into DogmaModeler so that the correctness of the built ontology can be checked by methods of Logical reasoning.

The initial and primitive results of this research appeared first in [HJ10], and then it was revised, extended, evaluated, and implemented in an article submitted to Data and Knowledge Engineering Journal, Elsevier on December 2011.

1.4 Overview of the thesis

The thesis is divided into six chapters. The current chapter provides an introduction, aim and objectives for this research. Chapter two introduces and defines ORM, Description Logics, and OWL 2 in addition to providing a thorough review of related work. Chapter three discusses the mapping from ORM into OWL 2; it also describes definitions, requirements, and works of mapping. Chapter four includes extending ORM for complete representation of OWL 2. Chapter five evaluates the OWL 2 mappings of ORM in addition to our ORM extension. Chapter six describes the implementation of our work in DogmaModeler tool. Chapter seven concludes the work.

Chapter Two

Background and Related Work

2.1 Introduction

In this chapter, we shed the light on background and related work. For background we highlight three fundamental related topics, namely, Object-Role Modeling (ORM), Description Logics, and Ontology Web Language (OWL 2) that are briefly discussed in the following related subsections with an example on the Object-Role Modeling (ORM) approach. For related work we briefly discus two themes and compare them to our thesis work, one of these themes consider the problem of ontology modeling a problem of visualization and the other develop formal semantics (i.e., formalize) such as UML and EER.

2.2 Object Role Modeling (ORM)

Object Role Modeling (ORM) which was introduced in the mid of 1970s, is used to model, transform and query about business information in a fact-oriented context. All facts and rules can be verbalized in natural languages that are understood of interested users including unknowledgeable technical users. ORM is an attribute-free modeling language, unlike Entity Relationship (ER) modeling and Unified Modeling Language (UML) class diagramming, where ORM treats all elementary facts as relationships between objects making implemented structures of ORM irrelevant to business semantics. The fact that ORM is an attribute-free modeling tool maintains the semantic of modeled domain, enhances semantic stability and makes it easy for ORM structures (of grouping facts) to be verbalized.

In order to build a needed ontology we can graphically use ORM notations to implement the conceptual modeling techniques in a reasonable way [GH08, JDM03, CH05].

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. As mentioned earlier, ORM has been used commercially for more than thirty years as a database modeling methodology and has been recently becoming popular not only for ontology engineering but also as a graphical notation in other areas such as modeling of business rules, XML schemes, data warehouses, requirements engineering, web forms, etc. ORM has an expressive and stable graphical notation. It supports not only n-ary relations and reification, but a fairly comprehensive treatment of many 'practical' and 'standard' business rules and constraint types such as mandatory, uniqueness, identity, exclusion, implications, frequency occurrences, subsetting, subtyping, equality, and many others [H89, H01, H04b, J07a, J07b].

ORM makes it easy to simplify the presented conceptual model using both natural language (via its verbalization capabilities) and graphical notations to present facts in their simple or elementary forms. In addition, ORM diagrams can be populated by examples to measure the correctness of the design [H01, H04b]. Practical use cases have shown that skills and know-how of using ORM can be acquired easily and in a short period of time even by non-IT specialists [J07a, JDM03]. Moreover, several modeling tools support ORM notation such as: Microsoft Visio ™, DogmaModeler, and Norma.

The example in Fig. 2.1.a below depicts a sample ORM diagram including several rules and constraints that ORM is capable of expressing graphically. Note the three basic constructs of ORM; object types, value types, and roles (forming relations).Object types are represented as solid-line ellipses, value types are presented as dashed-line ellipses and relations as rectangles, where one or more ORM roles for each ORM relation. For example, the relation (*WorksFor/Employs*) in Fig. 2.1.a is a binary relation (i.e., composed of two roles).

The preceding knowledge that underpinning ORM is NIAM (Natural Language Information Analysis Method) [H89]. NIAM was introduced in the early 70's. An important reason under designing NIAM is to use natural language to declare the "semantics'" of a business application's data. Fig. 2.1b shows the verbalization of the ORM rules presented graphically in

Fig. 2.1a. One of the most powerful features of ORM is its verbalization capability in which ORM diagrams can be automatically verbalized into pseudo natural sentences. In other words, all rules in a given ORM diagram can be translated into fixed syntax sentences [H01, JDM03, H04a, HC06 JKD06]. For example, the Mandatory constraint (•) between 'Person' and 'Has Gender' is verbalized by rule-1 in Fig.1b as "Each Person Has at least one Gender". Similarly, the role uniqueness constraint (↔) is verbalized by rule 2, Subtype (→) by rule 3, (↕) by rule 6, (⊙) and (⊗) between subtypes by rules 4 and 5, and between roles by rules 7 and 8. The value constraint ({'M','F'} on Gender) is verbalized by rule 9. These verbalizations are generated automatically by the DogmaModeler tool through using verbalization templates parameterized over a given ORM diagram. DogmaModeler enables verbalization in 11 different human languages [JKD06]. The main purpose of this verbalization is to simplify the communication with non-IT specialists and to allow them to better validate and build ontology models.

1. Each Person Has at least one Gender. (Mandatory)
2. Each Person Has at most one Gender. (Role uniqueness)
3. Each Male is a Person. Each Female is a Person. (Subtype)
4. Each Person cannot be a Male and a Female at the same time. (Exclusive)
5. Each Person must be, at least, Male or Female. (Totality)
6. If a Person is AffiliatedWith a Company then this Person WorksFor that Company. (Subset)
7. Same Car cannot be OwnedBy by Person and OwnedBy a Company at the same time. (Exclusion)
8. Each Car should be OwnedBy by Person or OwnedBy a Company, or both. (Disjunctive Mandatory)
9. A Gender can only be one of {M,F}. (Value Constraint)

(a)   (b)

Figure 2.1: Sample ORM Diagram along with the verbalization of its rules.

2.3 Description Logics (ALC, SHOIN and SROIQ)

Description logics [BCM+07] are a family of knowledge representation formalisms. Description logics are decidable fragments of first-order logic, associated with a set of automatic reasoning procedures [HST99]. The basic primitives of 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 $HumanMother \sqsubseteq Female \sqcap \exists HasChild.Person$. All DLs implies the open world assumption [NB02].

ALC [HST06] is simple description logic stands for Attributive Concept Language with Complements. Each feature of description logic has a letter assigned to it for example ALC is AL augmented with complements. ALC syntax is as follows, where N_C, N_R and N_O are atomic concepts. The followings are concepts: $\top$, $\bot$ and every $A \in N_C$ (all atomic concepts are concepts). If C and D are concepts and R is a role (binary relation) then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall R.C$, $\exists R.C$ are concepts.

We can use the constructs (primitives of the notations of concept and relationship) of ALC to represent the complex concept and relationship expressions. As an example the expression Person $\sqcap$ Doctor (Person and Doctor are atomic concepts) means persons that are also doctors. The expression Doctor $\sqcap$ $\forall$hasPatient.Person (hasPatient is an atomic role) means that doctors have only persons as patients.

For the semantic of ALC's we need the notation of interpretation $I = (\Delta^I, .^I)$, where Δ^I is the non empty set which is the domain of I and $.^I$ which is the interpretation function used to map every individual to an element $a^I \in \Delta^I$, every atomic concept A to a subset of Δ^I and every role name to a binary relation(subset of Δ^I x Δ^I). The following definitions represent the semantic of *ALC* constructs:

$$
\begin{aligned}
\top^I &= \Delta^I \\
\bot^I &= \emptyset \\
(C \sqcap D)^I &= C^I \cap D^I \quad \text{(intersection)} \\
(C \sqcup \neg D)^I &= C^I \cup D^I \quad \text{(union)} \\
(\neg C)^I &= \Delta^I \backslash C^I \quad \text{(complement)} \\
(\exists R.C)^I &= \{x \in \Delta^I \mid y \in \Delta^I : \langle x,y \rangle \in R^I \text{ and } y \in C^I\} \quad \text{(exists restriction)} \\
(\forall R.C)^I &= \{x \in \Delta^I \mid \langle x,y \rangle \in R^I \text{ implies } y \in C^I\} \quad \text{(universal restriction)}
\end{aligned}
$$

Figure 2.2: Semantic of *ALC*

SHOIN which is the base of Web Ontology Language (OWL) is expressive and decidable description logic. SHOIN equals (ALC extended with transitive roles(R_+)) + role hierarchy (H)($R \sqsubseteq S$) + nominals (O) ($C \equiv \{a,b,c\}$) + inverse (I)($S \sqsubseteq R^-$) + number restrictions (N).

SROIQ Description Logic [HST06] compromises both features of expressivity and decidability. SROIQ is an extension of SHOIN which is the underlying description logic of

OWL [BHH+04]. However, the rise of the Semantic Web increased the need for a more featured and expressive Description Logic to author ontologies. As a result of this increasing demand, SROIQ was introduced with many new features (especially regarding expressivity) which led to adopting SROIQ as the underpinning logic for OWL 2.

A Description Logic knowledge base is composed of two components: a TBox and an ABox. The TBox contains intensional knowledge in the form of a terminology. TBox is built through declarations that describe general properties of concepts. The ABox contains extensional knowledge which is also called assertional knowledge. It is knowledge that is specific to the individuals of the domain of discourse (knowledge specific to the instances) [BCM+07].

SROIQ syntax can be defined 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 general concept inclusion. SROIQ also allows hierarchy of roles $(R \sqsubseteq S)$, transitivity of roles (R_+), and inverse of roles $(S \sqsubseteq R^-)$. In addition, SROIQ allows everything SHOIN allows, in addition to the following [HST06]:

i) **Disjoint roles:** most description logics do not support disjoint roles and this makes them unbalanced. SROIQ is said to be balanced because it allows the expressivity of disjoint between roles. For example, the roles *brother* and *father* should be declared as being disjoint.

ii) **Reflexive, reflexive and antisymmetric roles:** these constraints are useful when using ABox to represent individuals. E.g., the role *loves* should be declared as being reflexive (one can love himself), and the role hasSibling should be declared as being irreflexive (one cannot be the sibling of himself).

iii) **Negated role assertion:** Although most Abox formalisms allow for only positive role assertions, SROIQ allows for negated roles assertions as well. For example, such

statements can be found in a SROIQ Abox: $(Rami, Tony): \neg knows$, which means that Rami do not know Tony.

iv) **Role inclusion axioms:** these roles are of the form $R \circ S \sqsubseteq R$ and $S \circ R \sqsubseteq R$. For example, given the following two axioms (1) $owns \circ hasPart \sqsubseteq owns$, and (2) the fact that each car contains an engine $Car \sqsubseteq \exists hasPart.Engine$. This implies that an owner of a car is also an owner of an engine, i.e., the following subsumption is implied: $\exists owns.Car \sqsubseteq \exists owns.Engine$.

v) **Universal role** U.

vi) **Local reflexivity** of the form $\exists R.Self$. For example, the following expresses the fact that somebody loves himself/herself: $\exists likes.Self$.

vii) SROIQ also provides what is referred to as **Rbox** which contains all statements concerning roles.

SROIQ (D) consists of ALC (description logic) + role chains = SR, SR + O (nominals(closed classes)), SRO+I(inverse roles), SROI+Q(qualified cardinality restrictions) and SROIQ+D(data types). Since SROIQ is an extension of ALC (introduced above) we will introduce the constructs of SROIQ not present in ALC through the following table that shows syntax and semantic of SROIQ description logic.

Table 2.1 : Syntax and Semantic of *SROIQ* other than *ALC*

	Name	Syntax	*Semantic*		Name	Syntax	*Semantic*
S ***R***	Role chains	$R \circ S \sqsubseteq R$	$\forall x \forall y (\exists z((R(x,z) \wedge S(z,y)) \rightarrow R(x,y)))$	*S* *R*	Nominals	$\{a\}$	$\{a^I\}$
	Transitivity	R^*	$(R^I)^*$	$+$ *O*	Individual equality	$a = b$	$a^I = b^I$
	role hierarchies	$R \sqsubseteq S$	$C^I \subseteq D^I$		Individual inequality	$a \neq b$	$a^I \neq b^I$
SRO $+$ *I*							
inverse roles	R^-	$\{\langle x,y \rangle \mid \{\langle y,x \rangle \in R^I\}$					
SROI $+$ *Q*							
Qualified Number Restriction	$\geq n R.C$	$\{x \in \Delta^I \mid \#\{y \in \Delta^I \mid \langle x,y \rangle \in R^I$ and $y \in C^I\} \geq n\}$					
	$\leq n R.C$	$\{x \in \Delta^I \mid \#\{y \in \Delta^I \mid \langle x,y \rangle \in R^I$ and $y \in C^I\} \leq n\}$					
	$= n R.C$	$\{x \in \Delta^I \mid \#\{y \in \Delta^I \mid \langle x,y \rangle \in R^I$ and $y \in C^I\} = n\}$					

2.4 OWL 2 Web Ontology Language

The W3C-recommended Web Ontology Language (OWL) [BHH+04] is a knowledge representation language [MMH04] used to publish and share ontologies on the Web, where this ontology language includes formally defined meanings. While the underpinning description logic of OWL is SHOIN, OWL 2[2] (the new version of OWL) is based on SROIQ description logic. Roughly speaking, one can often view OWL 2 as SROIQ written in other syntaxes (XML, RDF/XML, etc). Among OWL 2 basic constructs are: *Class* (corresponds to a 'concept' in SROIQ or called 'object-type' in ORM), *Property* (corresponds to a SROIQ 'relationship' or an ORM 'role'), and *Object* (corresponds to a SROIQ individual/assertion).

It is worth also noting here that OWL 2 is supported by several semantic reasoners such as RacerPro 2, Hermit, Pellet and Fact++[3].

As of October 27, 2009, OWL 2 has been set as a W3C recommended as a standard for ontology representation on the Web. Classes, properties, individuals, and data values are all supported by OWL 2 and stored semantically on the Web. OWL 2 ontologies are primary exchanged as RDF documents, where these ontologies can be used with information written in RDF. In addition to XML/RDF syntax which is used to serialize and exchange OWL 2 ontologies, functional syntax is used to determine the structure of OWL 2 and OWL/XML syntax is used as an XML serialization for better interoperability. OWL 2 elements are identified by Internationalized Resource Identifiers (IRIs). It extends OWL 1 which uses Uniform Resource Identifiers (URIs) [MGH+09, GW09]. Every IRI must be absolute to be published internationally. OWL 2 increases expressive language power for properties.

The following are some of the new features that distinguish OWL 2 from its OWL predecessor [GW09]:

   i)   Syntactic sugar

---

[2] www.w3.org/2009/pdf/REC-owl2-overview-20091027.pdf

[3] http://owl.man.ac.uk/factplusplus/

- DisjointUnion states that if we have the classes A, B and C then class A is the union of classes B or C and at the same time class B and C are disjoint to each other, where no individual can be an instance of class B and C at the same time.

- DisjointClasses states that if we have a set of classes' constraint by the construct DisjointClasses then these classes are disjoint to each another, where an individual of one class of the set cannot be an instance of the other classes of the set.

- NegativeObjectPropertyAssertion (also we have NegativeDataPropertyAssertion) states that a given individuals are not related to the intended object property.

ii) New constructs for properties

- Self restriction: ObjectHasSelf states that if we have an object property such as likes related to a class person then the individuals of this class are related to themselves by the object property likes.

- Property Qualified Cardinality Restrictions: ObjectMinCardinality, ObjectMaxCardinality, and ObjectExactCardinality; and respectively DataMinCardinality, DataMaxCardinality, and DataExactCardinality allow asserting minimum, maximum or exact qualified cardinality restrictions for object and data properties.

- Reflexive, Irreflexive, and Asymmetric Object Properties [MGH+09].

- Disjoint Properties states that if two or more object properties restricted by DisjointObjectProperties construct then these object properties are exclusive to each others. Also the same for data properties restricted by DisjointDataProperties construct.

- Property Chain Inclusion states that if we have two or more object properties chained by the construct ObjectPropertyChain in a SubObjectPropertyOf axiom then as a result we have a property that is the composition of several properties.

- Keys: HasKey allows defining keys for a given class.

iii) Simple metamodeling capabilities

- Punning: OWL 2 allows using the same term for both a class and an individual at the same time.

iv) Extended Annotations

- Annotations: the construct AnnotationAssertion is used to assert annotations for classes, properties and individuals.

- Axioms about annotation properties: (AnnotationPropertyDomain) and ranges (AnnotationPropertyRange) and participate in an annotation property hierarchy (SubAnnotationPropertyOf). These special axioms have no semantic meaning in the OWL 2 Direct Semantics, but carry the standard RDF semantics in the RDF-based Semantics (via their mapping to RDF vocabulary).

v) Declarations: The prior announcement for an entity such as a class is important for error catching where a declaration in OWL 2 for an entity such as class, datatype, object property, data property, annotation property, or individual indicates that an entity is part of the terminology of ontology.

vi) Top and Bottom Properties: In addition to top and bottom classes (owl:Thing and owl:Nothing), OWL 2 provides top and bottom object properties(owl:topObjectProperty, owl:bottomObjectProperty) and data properties (owl:topDataProperty, and owl:bottomDataProperty).

vii) IRIs: OWL 2 uses Internationalized Resource Identifiers (IRIs) for identifying the elements of ontologies and ontologies themselves in order to come over the language limitations of URIs [MPP09, SWM04] that were used in OWL 1.

2.5 Related Work

As discussed previously, our work revolves around establishing ORM as a practical and methodological means for ontology engineering by combining its strengths with those of the Web Ontology Language (OWL). There are several approaches and tools similar to our work, which aim to use graphical notations such as UML and EER for ontology modeling. Also, in particular, some of them aim to model OWL ontologies graphically. Some of these approaches and tools consider the problem of ontology modeling a problem of visualization, thus ignoring the underpinning semantics. However, some efforts exist to develop formal semantics (i.e., formalize) UML and EER. In this section, we discuss these efforts and compare them to our approach.

Many efforts exist to use UML or UML-based notations for graphical representation of ontologies. Cranefield *et al, in* [CHD+02], proposed using UML as an ontology visualization and editing tool. Although their goal is to visualize and edit ontologies, they did not consider the underpinning semantics. Their work does not also provide any type of mapping of the graphical notation (i.e., UML) into OWL. Brockmans *et al,* in [BVEL04], introduced a UML-based notation for the visualization of OWL ontologies by developing a UML profile. In such approach, OWL is visualized based on the UML profiles of the Ontology Definition Metamodel (ODM[4]). ODM defines a set of UML metamodels and profiles for the development of RDF and OWL. These UML profiles adapt UML notations to provide a suitable visual representation of RDF and OWL ontologies. This representation of ontologies using ODM enables one to only visualize the ontology but does not capture the semantics of it. This is due to the challenges of developing well-formed and usable UML models with equivalent semantics in OWL. However, in [KBB+09], Kendall *et al* presented some potential extensions to the UML profiles of ODM to address some of the requirements of OWL 2. It is important to note also that UML itself is a very basic notation; as will be demonstrated later, ORM is much more expressive as it allows 20 types of rules to be expressed graphically, while UML support only cardinality rules.

---

[4] http://www.omg.org/spec/ODM/1.0/

Many ontology editing tools are available such as Protégé[5], TopBraid[6], NeOn[7] and GrOWL[KWF07]. These tools are used to build and edit ontologies by enabling the creation of classes, object properties and various constraints. After building the needed ontology, it can be visualized graphically. However, such tools do not capture the ontology semantics. In our methodology, we build the ontology graphically using ORM notation which is then mapped into the equivalent OWL 2 constructs capturing the semantics of the ontology.

Protégé is an open source ontology editor for OWL and recently OWL 2. Protégé is built using Java and it is a framework for other project plug-ins. By using protégé you can build the needed ontology by using all the constructs of OWL 2 based in SROIQ (D) description logic. One can create classes, object properties, data object properties, class expressions, individuals and the relations between these concepts using the various constraints of OWL 2. But here we still talk about building the intended ontology in a tree-based hierarchy methodology (composed of classes, properties and individuals) and not in a graphical context representation, besides not gaining the big picture of modeling. Protégé provides two main techniques of modeling ontologies, one is Protégé-OWL[8] editor, and the other is Protégé-Frames editor, based on Open Knowledge Base Connectivity protocol (OKBC). The built ontology can be visualized in the shape of tree indicating the classes and the relations between them, but here we are talking about just displaying the built ontology in a graph representation and not capturing the semantic of built ontology. OntoViz which is a visualization plug-in included in Protégé and uses the library of GraphViz[9] to create an ontology in 2D graph. The built ontology is displayed in 2D graph showing the classes, their properties, role relations and instances. Indeed our work is in opposite context where we can build the needed ontology graphically using the extended ORM. The extended ORM includes all the needed OWL 2 constructs for building the intended ontology. Once the ontology is built graphically in an easy way with capturing the semantic of ontology, it is automatically formalized into OWL 2 using the DogmaModeler tool. This highlights our wok in enabling the building of ontology graphically and in an easy way in comparison with the wide used Protégé modeling tool.

---

[5] http://protege.stanford.edu/
[6] http://www.topquadrant.com/products/TB_Composer.html
[7] http://neon-toolkit.org
[8] http://protege.stanford.edu/publications/ontology_development/ ontology101.pdf
[9] http://graphViz.org

TopBraid is a commercial modeling tool for developing ontologism in semantic web context. TopBraid Composer is an RDF (Resource Description Framework) and OWL editor, where one can create the RDF/OWL files and query over them using SPARQL. By using the TopBraid editor you can create and edit the needed classes, object properties and various constraints. After creating ontology you can display the graph representation of it, similar to that of Protégé. As we said before our work is opposite where you can create an ontology graphically using the extended ORM and consequently you will have its semantic equivalent in OWL 2.

NeOn is an open source tool that provides an ontology engineering environment based on Eclipse Integrated Development Environment (IDE). It includes many plug-ins that support many ontology engineering activities and provides a means of visualizing the ontology as it is being built.

GrOWL [KWF07] is also a tool for visualizing and editing OWL ontologies with advanced browsing and navigation tools. Our approach enables one to graphically build the ontology using ORM notation with semantic capturing and this Ontology is automatically mapped into OWL and validated.

Mapping EER is done in [BCM+07] and UML is done in [BCG05]. Formalizing form EER and UML into DLR_{idf} description logic is done in these two works [BCM+07, BCG05]. Jarrar [J07a] implemented mapping from ORM to SHOIN/OWL description logic. SHOIN is chosen because of its ability of expressiveness and decidability. Each rule of ORM that is supported by SHOIN is mapped into SHOIN. Twenty two cases of ORM constructs are mapped, where the purpose is to use ORM as a technique and expressive notation for ontology engineering. Although mapping ORM into SHOIN is achieved in [J07b], but mapping ORM into OWL is not achieved. In our research, mapping between ORM and OWL 2 is achieved and is implemented automatically by DogmaModeler tool.

It is also worth noting that the ICOM tool was one of the first tools to enable automated reasoning with conceptual modeling. ICOM [FN00] is a conceptual modeling tool, which is used for knowledge representation by allowing one to design multiple extended ontologies. ICOM was first released in 2000 [FN00], but recently it has version 3 release [FFT10]. ICOM has reasoning capabilities. ICOM supports ontology modeling using a graphical notation that is a mix of UML and EER notations. ICOM[10] enables one to build ontologies using multiple ER or UML class diagrams with various inter and intra constraints. The class diagrams and constraints are formally translated into class-base logic. It can express EER and UML class diagrams enriched with various constrains, into classes and relations involved with expressions, representing various ontologies. ICOM is integrated with a description logic reasoning server that acts as a background inference engine to enable automatic reasoning.

ICOM does not express ORM, where our work express extended ORM for full representation of OWL 2, where ORM is friendlier for use than ER and UML and at the same time more rich with needed constrains for knowledge representation. ICOM represents ER or UML class diagrams into DIG for the purpose of reasoning, but not into OWL 2, where in our work we map extended ORM into OWL 2. The implementation of mapping is done using the extended DogmaModeler (extended to hold mapping between ORM and OWL 2). Once we build the needed ontology graphically using ORM under DogmaModeler tool. An OWL 2 file is automatically generated. We can reason about the generated OWL 2 file by using Hermit reasoned(integrated with DogmaModeler) that supports OWL 2.

Bārzdiņš et al. [BBC+10] proposed a hard extension of UML class diagrams to visualize OWL 2 in a dense way, but here we still talking about visualization method that not exactly capture the semantics of OWL 2. Here textual representation is still used to represent OWL 2 constructs in a graphical way like disjoint, equivalent and others, so here the representation of OWL 2 constructs is not graphically pure while our work semantically represents OWL 2 constructs with pure graphic notations. The proposed extension is implemented to visualize OWL 2 using a UML graphical editor called OWLGrEd. As we said, in our work we are talking about extending ORM to graphically represent the constructs of OWL 2 that are not

---

[10] www.inf.unibz.it/~franconi/icom/files/icommanual.pdf

represented by ORM, in order to be able to use ORM as an interface of OWL 2 in order to be able to build an ontology graphically in an easy way . This extension is implemented using the extended tool (DogmaModeler). Some of the extension that is done in [BBC+10] to represent uncovered OWL 2 constructs by UML is already included in ORM like equivalent and disjoint properties, in addition to the Boolean relations between class and class expression or even between class expressions themselves. OWL 2 is indicated by its expressive power of class expressions and at the same time ORM graphically do represent the relation between class and class expression, as also between class expressions themselves. Where ORM as a basis of concepts and relations between them in addition to the used constraints graphically represents the class expressions like denoting that a class A is a subclass of an expression of 'some values from' or 'all values from'. All these features indicate that ORM is a robust graphical tool as a fact oriented conceptual modeling tool and contains a rich set of constraints not covered in UML and ER.

Many other similar tools and frameworks which enable a text tree-based methodology for knowledge representation and not enabling graphical building of ontology are available such as, IBM Integrated Ontology Development Toolkit [IIODT[11]], SWOOP[12], DOME, and DERI[13] Ontology Management Environment among many others. Although all of these tools allow a graphical representation of the ontology, this representation is merely a visualization that ignores the underpinning semantics, in contrast to our proposed ORM ontology engineering paradigm.

The investigation of the ORM notation is done by formalizing it using both DLR_{idf} description logic [J07a, K07] and SHOIN description logic [HJ10, J07b]. The main purpose of this formalization/mapping was to enable automated reasoning on the formal properties of ORM diagrams, such as detecting constraint contradictions and implications. Thus, in this study of ORM, the native semantics of ORM were used which were expressed in DLR and SHOIN Description Logics. Anoher point we are concerning about SROIQ which is the logic underpinning OWL 2 that is decidable and supported by many reasoners and not DLR_{idf} as in

---

[11] http://www.alphaworks.ibm.com/tech/
[12] http://www.mindswap.org/2004/SWOOP/
[13] http://dome.sourceforge.net/

[K07]. In [BMST07] an attempt of mapping OWL into ORM/RIDL was done but it was not complete where not all ORM constructs were covered in this research as Ring constraints and role disjoitness; In addition the author did not rely in specified description logic and relied in OWL and not OWL 2 that includes new features where OWL 2 was not available at that time of research. However, in the mapping presented in this thesis, we have altered the native semantics of ORM to adapt it to that of OWL 2 (i.e., we have used the semantics of OWL 2 not of ORM). That is, we have expressed the semantics of OWL 2 graphically using the ORM notation, without employing the semantics of ORM. For example, ORM subtypes are proper subtypes. 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. However, such loops according to the semantics of OWL 2 are allowed which means that the classes involved in the loop are equivalent. Furthermore, object types (i.e., classes) in ORM are mutually exclusive. That is, according to ORM semantics, it is not allowed for an object-type to be a subtype of two different object-types, unless these two supertypes have a common supertype. However, such a multiple inheritance is allowed according to the semantics of OWL 2.

Another important difference between ORM and OWL 2 semantics is that ORM adopts a close world assumption while OWL 2 adopts an open world assumption [NB02]. A closed world assumption states that any statement that is not known to be true or false is false. On the contrary, an open world assumption states that, unless the truth-value of a statement is explicitly determined, it is unknown. For instance, originally, ORM assumes that any two object types are disjoint, without the need to state it explicitly (closed world assumption). Here, we follow OWL 2's open world assumption where any two object types are not known whether they are disjoint or not, except if stated explicitly. The same applies, for example, to instances (assertions), where OWL 2 semantics states that any two instances are not considered different (unequal) or equal unless it is explicitly stated (open world assumption). In short, in this thesis, we don't present or follow ORM semantics, but rather use the ORM graphical notation to depict OWL 2 constructs using OWL 2 semantics.

Chapter Three

Mapping ORM into SROIQ/OWL 2

3.1 Introduction

Since we concentrate on the ability of expressivity and decidability for our mapping results (SROIQ achieves this ability [HST06]), we will use SROIQ Description Logic (which is the most common in ontology engineering and it is the language underpinning OWL 2) as a reference to map from ORM into OWL 2. First, we formally map the ORM construct into SROIQ Description Logic and then we represent this model in OWL 2. Our scope of conversion is every construct of ORM.

3.2 Use Case

Before delving into the details of the mapping of ORM using SROIQ/OWL2, we first present a use case where we map the sample ORM diagram in Figure 2.1 into OWL2. Part of the mapping is depicted in Figures 3.1 through 3.3 along with a brief discussion. The full mapping using OWL/XML is provided in Appendix A-1.

The basic ORM constructs are mapped to OWL 2 as follows. An object/value type is mapped as a *Class* in OWL 2 whereas an ORM role is mapped as an *ObjectProperty/DataProperty*. Before mapping the ORM rules and constraints to OWL 2, one must first declare the object/value types and roles in OWL 2. Fig. 3.1 depicts the declarations of the object types *Person* and *Company* and the ORM relation *WorksFor/Employs* of Fig. 3.2.a. The complete declarations of the ORM diagram are provided in the Appendix.

```
1.  <Declaration>                      7.  <Declaration>
2.     <Class IRI="#Person"/>          8.     <ObjectProperty IRI="#Employs"/>
3.  </Declaration>                     9.  </Declaration>

4.  <Declaration>                      10. <Declaration>
5.     <Class IRI="#Company"/>         11.    <ObjectProperty IRI="#WorksFor"/>
6.  </Declaration>                     12. </Declaration>
```

Figure 3.1: OWL 2 declarations of Classes and Object Properties

The diagram in Fig. 3.2.a contains 9 rules which we map into SROIQ/OWL-2 in Fig. 3.2.b below. These rules are: *(1) Subsumption (subtype)*, *(2) Mandatory*, *(3) Role Uniqueness*, *(4) Total Constraint*, *(5) Exclusive Constraint*, *(6) Subset Constraint*, *(7) Disjunctive Mandatory (inclusive-or)*, *(8)Exclusion*, and *(9)Value Constraint*.
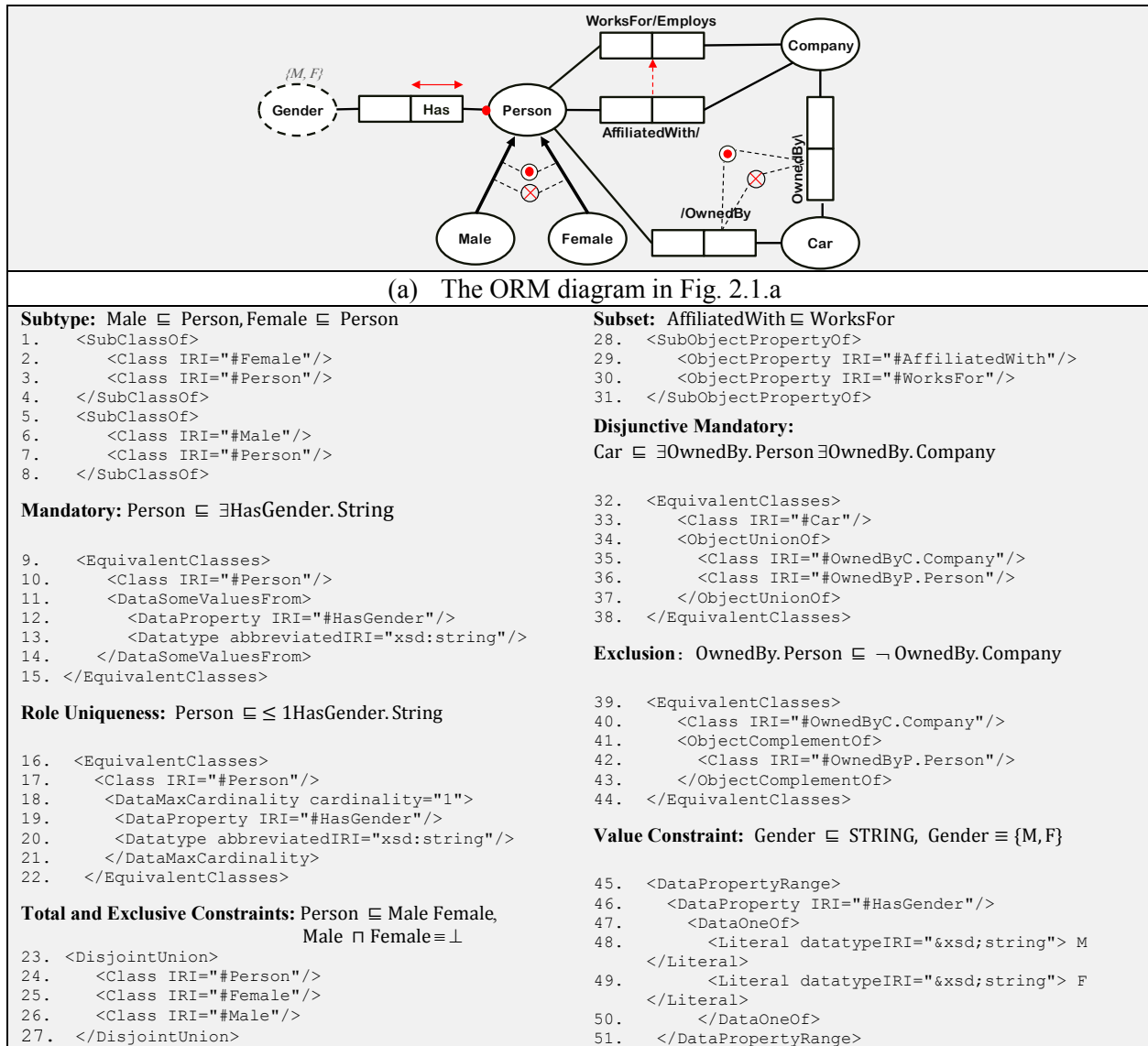

(a)   The ORM diagram in Fig. 2.1.a

Subtype: Male $\sqsubseteq$ Person, Female $\sqsubseteq$ Person

```
1.  <SubClassOf>
2.     <Class IRI="#Female"/>
3.     <Class IRI="#Person"/>
4.  </SubClassOf>
5.  <SubClassOf>
6.     <Class IRI="#Male"/>
7.     <Class IRI="#Person"/>
8.  </SubClassOf>
```

Mandatory: Person $\sqsubseteq$ $\exists$HasGender. String

```
9.   <EquivalentClasses>
10.      <Class IRI="#Person"/>
11.      <DataSomeValuesFrom>
12.        <DataProperty IRI="#HasGender"/>
13.        <Datatype abbreviatedIRI="xsd:string"/>
14.      </DataSomeValuesFrom>
15.  </EquivalentClasses>
```

Role Uniqueness: Person $\sqsubseteq$ $\leq$ 1HasGender. String

```
16.   <EquivalentClasses>
17.     <Class IRI="#Person"/>
18.     <DataMaxCardinality cardinality="1">
19.       <DataProperty IRI="#HasGender"/>
20.       <Datatype abbreviatedIRI="xsd:string"/>
21.     </DataMaxCardinality>
22.    </EquivalentClasses>
```

Total and Exclusive Constraints: Person $\sqsubseteq$ Male Female, Male $\sqcap$ Female $\equiv$ $\bot$

```
23. <DisjointUnion>
24.    <Class IRI="#Person"/>
25.    <Class IRI="#Female"/>
26.    <Class IRI="#Male"/>
27. </DisjointUnion>
```

Subset: AffiliatedWith $\sqsubseteq$ WorksFor

```
28. <SubObjectPropertyOf>
29.    <ObjectProperty IRI="#AffiliatedWith"/>
30.    <ObjectProperty IRI="#WorksFor"/>
31. </SubObjectPropertyOf>
```

Disjunctive Mandatory:
Car $\sqsubseteq$ $\exists$OwnedBy. Person $\exists$OwnedBy. Company

```
32.  <EquivalentClasses>
33.     <Class IRI="#Car"/>
34.     <ObjectUnionOf>
35.       <Class IRI="#OwnedByC.Company"/>
36.       <Class IRI="#OwnedByP.Person"/>
37.     </ObjectUnionOf>
38.  </EquivalentClasses>
```

Exclusion: OwnedBy. Person $\sqsubseteq$ $\neg$ OwnedBy. Company

```
39.  <EquivalentClasses>
40.     <Class IRI="#OwnedByC.Company"/>
41.     <ObjectComplementOf>
42.       <Class IRI="#OwnedByP.Person"/>
43.     </ObjectComplementOf>
44.  </EquivalentClasses>
```

Value Constraint: Gender $\sqsubseteq$ STRING, Gender $\equiv$ {M, F}

```
45.  <DataPropertyRange>
46.    <DataProperty IRI="#HasGender"/>
47.     <DataOneOf>
48.       <Literal datatypeIRI="&xsd;string"> M
       </Literal>
49.       <Literal datatypeIRI="&xsd;string"> F
       </Literal>
50.     </DataOneOf>
51.   </DataPropertyRange>
```

(b)   The SROIQ/OWL 2 mapping of the ORM rules in (a)

Figure 3.2: The ORM diagram of Fig. 2.1.a and the mapping of its rules to SROIQ/OWL 2

Subsumption (subtype) is depicted as an arrow (→) in ORM. In our example, this arrow is seen in two places: between *Male* and *Person*, and between *Female* and *Person*. This means that all instances of *Male* (all males) form a subset of the population of *Person* (all persons). This is also true for the object-type *Female*. In OWL 2, *SubClassOf* construct is used to represent this rule (lines 1-8, Fig. 3.2.b).

Mandatory is depicted as a dot (●) on the line connecting *Person* object type with *hasGender* role. This constraint indicates that, in every interpretation of this schema, each instance of the object-type *Person* must have at least one *Gender*. This rule is mapped in OWL 2 in lines 9-15, Fig. 3.2.b.

Role Uniqueness is depicted by an arrow ↔ spanning along the role *hasGender*. In OWL 2, we map this rule as shown in lines 16-22 in Fig. 3.2.b. This constrains indicates that, in every interpretation of this schema, each instance of the object-type *Person* must have at most one *Gender*.

Total and Exclusive Constraints are depicted as (⊙) and (⊗) between the *Male* and *Female* subtypes. The Total Constraint means that the population of *Person* is exactly the union of the population of *Male* and *Female* subtypes. The exclusive constraint means that the intersection of the population of *Male* and *Female* is always empty. In our example, we use *DisjointUnion* OWL 2 construct (lines 23-27, Fig. 3.2.b) which expresses both Total and Exclusive constraints.

Subset Constraint is depicted in the ORM diagram in Fig. 3.2.a as an arrow (↕) between the roles *AffiliatedWith* and *WorksFor*; which means that the role *AffiliatedWith* is a subset of role *WorksFor*. That is, all Persons who are *affiliated with* a Company must *work for* that company. This is written in SROIQ as: AffiliatedWith ⊑ WorksFor. Representation in OWL 2 is done using *SubObjectPropertyOf* (lines 28-31, Fig. 3.2.b)

Disjunctive Mandatory (inclusive-or), is depicted as (⊙) between two or more roles, illustrating that the disjunction of these roles is mandatory for members. In our example, each instance of object-type *Car* must be owned by at least a Person or a Company or both. This is

written in SROIQ description logic as: Car ⊑ ∃OwnedBy. Person ⊔ ∃OwnedBy. Company). This rule is mapped in OWL 2 in lines 32-38.

Exclusion, is represented as (⊗) between the roles it connects (in Fig. 3.a, it connects between the two *OwnedBy* roles of object-type *Car*). It means that each member of the population cannot play both roles constrained by exclusion. In the example, no car can be owned by a Person and a Company at the same time (OwnedBy. Person ⊑ ¬ OwnedBy. Company). Representation in OWL 2 is depicted in Fig 3.2.b (lines 39-44).

Value Constraint, is represented as {'M','F'} above the *Gender* dashed-line ellipse. This constraint indicates the possible set of values that the value type *Gender* can be populated with. Here, *Gender* can take any of the two STRING values: 'M' and 'F'. No other value is allowed. This constraint is mapped in OWL 2 in lines 45-51.

In the following subsections we thoroughly discuss the mappings of the ORM constructs into OWL 2 and its underpinning SROIQ Description Logic.

3.3 Object-Types and relations

3.3.1 Binary and N-ary relationships ($n \geq 1$):

ORM supports *n*-ary relationships, where $n \geq 1$. Each argument of a relationship in ORM is called a role. For example, consider the binary ORM relationship *WorksFor/Employs* in Fig. 3.2.a which has two roles, namely, *WorksFor* and *Employs*. However, SROIQ only supports binary relationships. Note that an ORM *role* is represented as a *relationship* in SROIQ. Thus, a binary ORM relationship is represented by two SROIQ relationships (that represent ORM roles) in addition to an inverse axiom to state that both SROIQ relationships are inverse to each other. Fig. 4.a (Rule-1) depicts a binary ORM relationship, its formalization into SROIQ, and its mapping into OWL 2. Fig. 3.3.b (Case-1) shows the general case of an ORM *n*-ary relationship, which cannot be represented in SROIQ/OWL 2. One can refer to [J07a] for the representation of Case-1 using DLR Description Logic. In this case, however, the n-*ary* relationship can be converted to binary relationships [H01] and then mapped into SROIQ/OWL 2.

The mapping of the binary ORM relation presented in Rule-1 represents the case where *A* and *B* (Fig. 3.3.a) are both object-types (equivalent to OWL 2 classes). However, in the case where either *A* or *B* are ORM value-types, the value-type is mapped into a "Literal" (the universal datatype) in OWL 2, or to any of its sub-datatypes. If, for example, *B* is a value-type, as depicted in Rule-1', the ORM role r_A is mapped into an OWL 2 *DataProperty* with datatype "Literal". Notice that this *DataProperty* is called r_AB; the concatenation of the names of both the ORM role (r_A) and the ORM value-type (B). Also note that we don't need an additional inverse axiom, because in this case only one *DataProperty* is needed to represent the ORM relation between an object-type and a value-type.
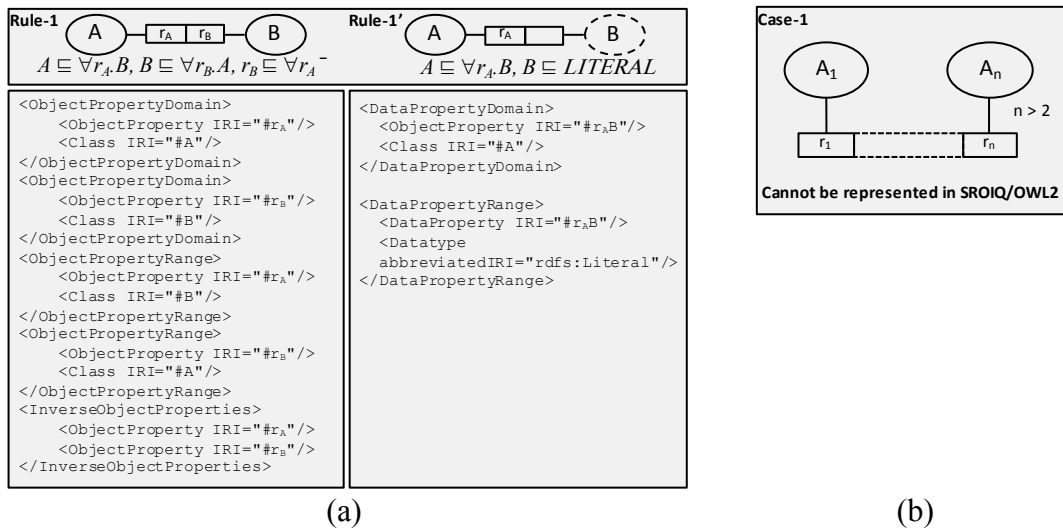

Figure 3.3: Binary and N-ary relationship

3.3.2 Unary Relationship:

Although unary roles are allowed in ORM, they cannot be represented directly in SROIQ/OWL 2. The example below (Fig. 3.4.a) shows an ORM unary relationship which means that a person may smoke; or in FOL [J07b]: $\forall x \ (Smokes(x) \rightarrow Person(x))$. The population of this fact is either true or false. In order to map ORM unary roles into SROIQ/OWL 2, 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 SROIQ/OWL 2, where the second concept is BOOLEAN. Rule-2 in Fig. 3.4.b presents the general case mapping of ORM unary fact types to SROIQ/OWL 2.
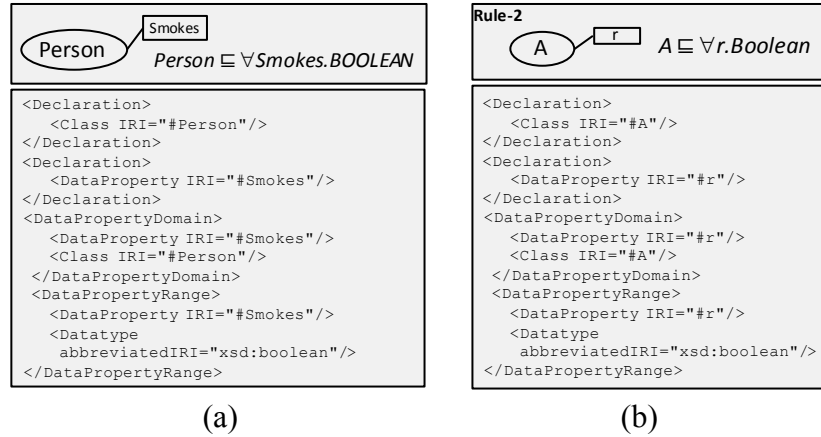
Figure 3.4: Unary Relationship

3.4 Subtypes

ORM subtypes are proper subtypes. That is, as discussed earlier, 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 $A \neq B$, i.e., loops are illegal in ORM. However, because the focus of this thesis is to establish a graphical representation of OWL 2, we adopt the semantics of OWL 2, whose subtype relation, *SubClassOf*, is not a *proper* subtype (i.e., loops are allowed). Thus, for example, the axiom "Female is a Person" is written using OWL 2 semantics as: $Woman \sqsubseteq Person$, without the need to add the axiom ($Person \not\sqsubseteq Woman$), as opposed to following ORM semantics. Rule-3 presents the mapping of the general case of subtypes using SROIQ/OWL 2.

3.5 Total Constraint

Total constraint ($\odot$) in ORM is equivalent to *UnionOf* construct in OWL 2. This rule means that the population of the supertype is exactly the union of the population of all subtypes constrained by this rule. Rule-4 represents the formalization of the general case.

3.6 Exclusive Constraint

ORM exclusive constraint ($\otimes$) is equivalent to *DisjointClasses* construct in OWL 2. It means that the population of the subtypes constraint by this rule is pairwise distinct, i.e., the intersection of the population of each pair of the subtypes must be empty. Rule-5 represents the formalization of the general case of the exclusive constraint.

Please note that in most of the examples and general case mappings in this thesis, the OWL 2 declarations are omitted due to space limitations.
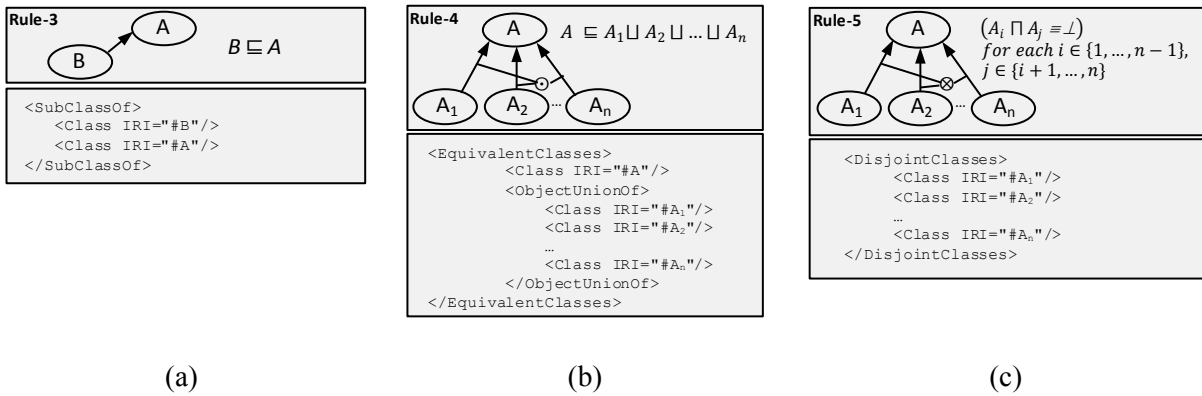
(a)   (b)   (c)

Figure 3.5: Formalization of ORM Subtype, Total Constraint and Exclusive Constraint.

3.7 Mandatory Constraints

3.7.1 Role mandatory:

ORM's Role mandatory constraint is depicted as a dot on the line connecting a role with an object type. Rule-6 of Fig. 3.6.a presents the general case formalization of this rule. Each instance of the object-type A must be related to at least one instance of object-type B by the relation r_A/r_B. In OWL 2, this rule is mapped using *ObjectSomeValuesFrom/ DataSomeValuesFrom* constructs as shown in Fig. 7.a. *ObjectSomeValuesFrom* is used when B is object-type, where as *DataSomeValuesFrom* is used when B is value-type. OWL 2 qualified minimum cardinality (*ObjectMinCardinality/DataMinCardinality)* construct can also be used to restrict the population of A to at least relate with one instance of B. However, the usage of *ObjectSomeValuesFrom/ DataSomeValuesFrom* is more elegant than *MinCardinality*.

3.7.2 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. In the general case presented in Fig. 3.6.b along with its formalization, each instance of object-type A must play at least one of the constrained roles: $(r_1,...,r_n)$ related to the instances of classes $A_1, ... A_n$ respectively.
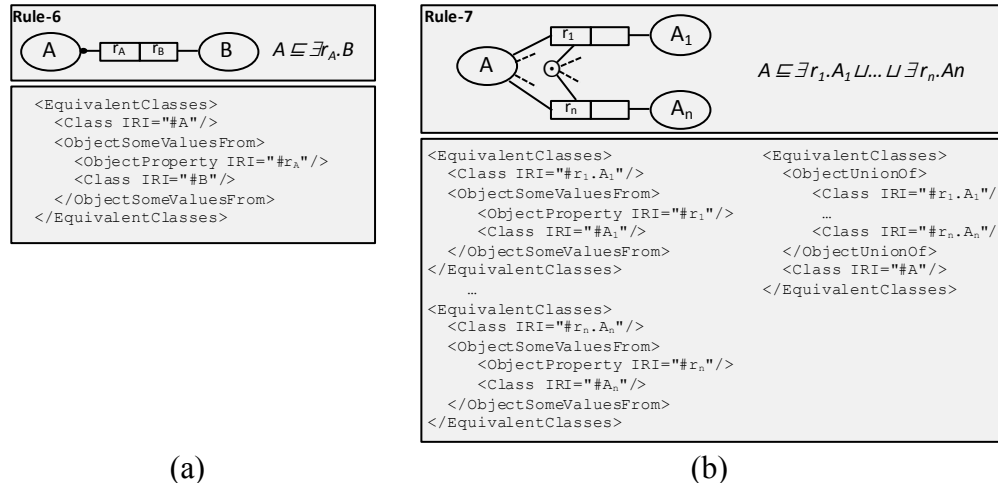
(a)   (b)

Figure 3.6: Mapping of ORM Mandatory Constraints

3.8 Uniqueness Constraints

One can distinguish between three types of Uniqueness Constraints in ORM, namely, role uniqueness, predicate uniqueness, and external uniqueness.

3.8.1 Role Uniqueness:

Role uniqueness is represented by an arrow spanning a single role in a binary relationship. Rule-8 of Fig. 3.7.a presents the general case mapping of this rule. This constraint means that each instance of an object-type A plays the relation r_A for at most one instance of B. Role uniqueness is mapped to OWL2 using qualified maximum cardinality (*ObjectMaxCardinality/DataMaxCardinality)* construct (restricted by '1').

3.8.2 Predicate Uniqueness:

This constraint is represented in ORM, as shown in Fig. 3.7.b, by an arrow spanning more than a role in an *n*-ary relationship. In the example shown in Fig. 3.7.b, in any instance of this relation, both *Student* and *Vehicle* must be unique together, i.e., functional dependency. Although this constraint can be represented using First Order Logic (FOL) and *DLR$_{idf}$* Description Logic [J07a, J07b], it cannot be represented using SROIQ/OWL 2.

3.8.3 External Uniqueness:

As shown in Fig. 3.7.c, ORM External Uniqueness constraint (denoted by 'U'), applies to roles from different relationships. The roles that participate in such a uniqueness constraint uniquely refer to an object-type. For example (Fig. 3.7.c), the combination of (Author, Title,

Edition) must be unique, i.e., different values of (Author, Title, Edition) refer to different books. This constraint cannot be represented using SROIQ/OWL 2.
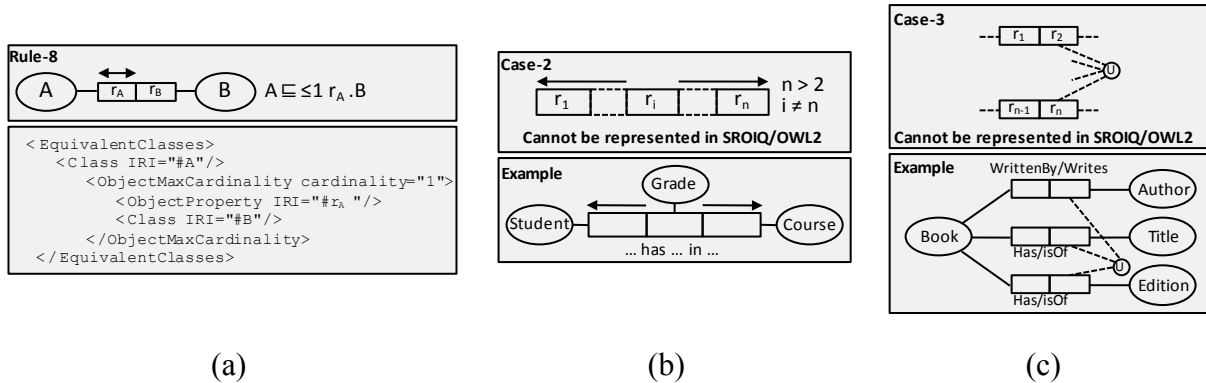

(a)   (b)   (c)

Figure 3.7: Uniqueness Constraints

3.9 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.

3.9.1 Role Frequency:

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. Fig. 3.8.a depicts the general case formalization of this rule. This constraint means that role r_A is played by the object-type A for a number of occurrences between *n* and *m*. We map this constraint to OWL2 by using the qualified number restrictions of OWL 2 *ObjectMinCardinality/DataMinCardinality* and *ObjectMaxCardinality/DataMaxCardinality* constructs.

3.9.2 Multiple-Role Frequency:

A multiple-role frequency constraint spans more than one role (Figure 3.8.b). This constraint means that, in the population of the constraint relationship, the constraint roles must be played together by the related object-types for a number of occurrences between *n* and *m*. Multiple-role Frequency Constraint cannot be formalized in Description Logic [J07b] and OWL 2.
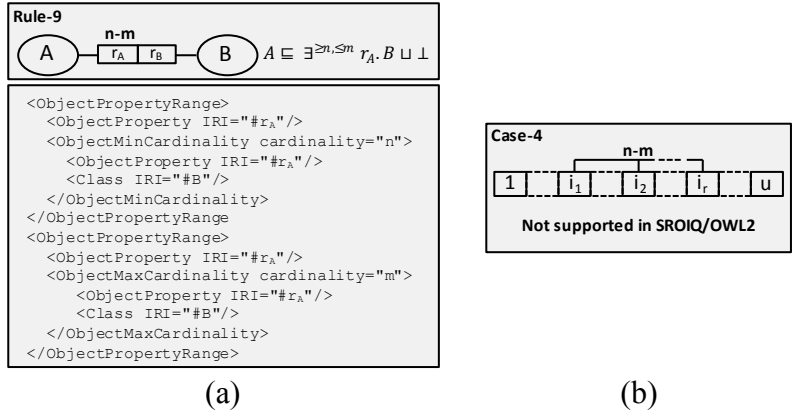
Figure 3.8: Frequency Constraints

3.10 Value Constraint

The value constraint in ORM indicates the possible set of values (i.e., instances) that an object-type can be populated with. A value constraint on an object-type A is denoted as a set of values $\{x_1, x_2, …, x_n\}$ depicted near an object type. Value constraints can be declared only on ORM Lexical Object Types (LOT), which are depicted as dotted-line ellipses, and the values should be well-typed, i.e., their data types should be either *String* such as {'hi', '98', 'it'} or *Number* such as {3,4,5}. Notice that quotes are used to distinguish string values from number values. It is worth noting that OWL 2 supports many data types besides integer and string which is an advantage of OWL 2 over OWL 1 (which only supports integers and strings). OWL 2 *DataOneOf* construct is used to map the Value constraints (Fig. 3.9).
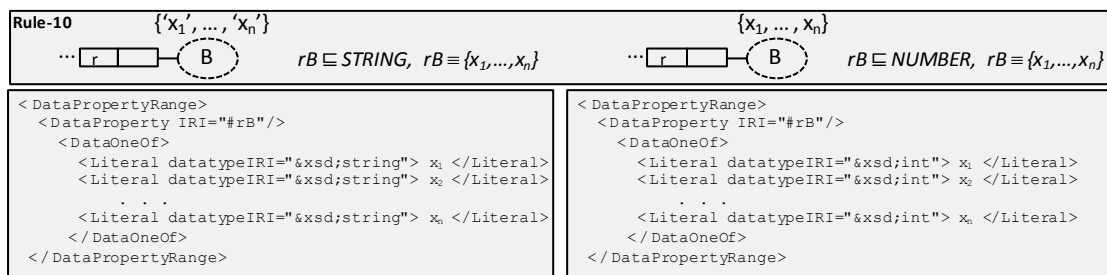

Figure 3.9: Value Constraints

3.11 Subset Constraint

The subset constraint ($\rightarrow$) between two roles is used to restrict the population of these roles so as one is a subset of the other. Fig. 3.10.a (Rule-11) depicts the general case mapping into SROIQ/OWL 2. It shows that all instances of A which plays the role *'s'* must also play the role

'r'. Rule-12 formalizes the case of subset constraint between two relations: the set of tuples of the subsuming relation is a subset of the tuples of the subsumed relation.

ORM also allows subset constraints between tuples of roles (not necessarily contiguous) as shown in case-5, 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. However, this last case cannot be represented in SROIQ/OWL 2.
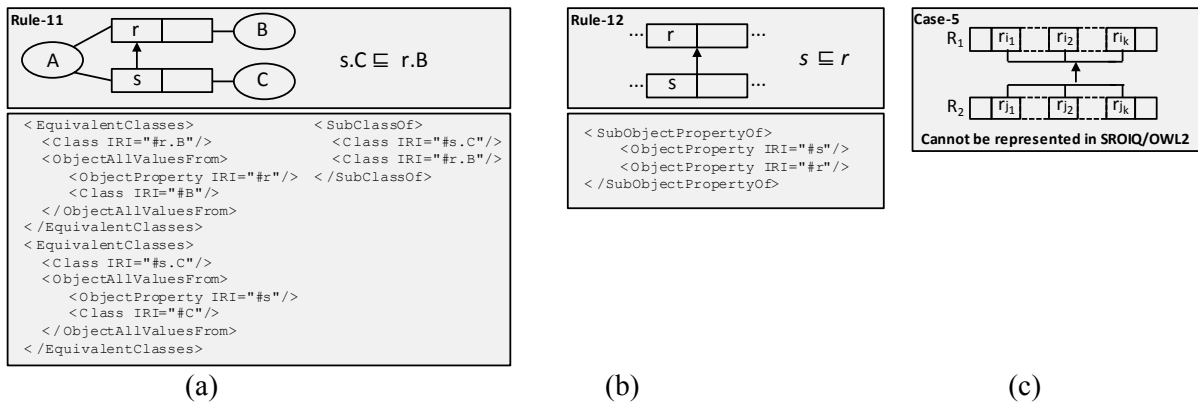

(a)   (b)   (c)

Figure 3.10: Subset Constraints

3.12 Equality Constraint

Similar to the subset constraint, the equality constraint ($\leftrightarrow$) between roles and relations are mapped as shown in rules 13 and 14 (Fig. 3.11) respectively.
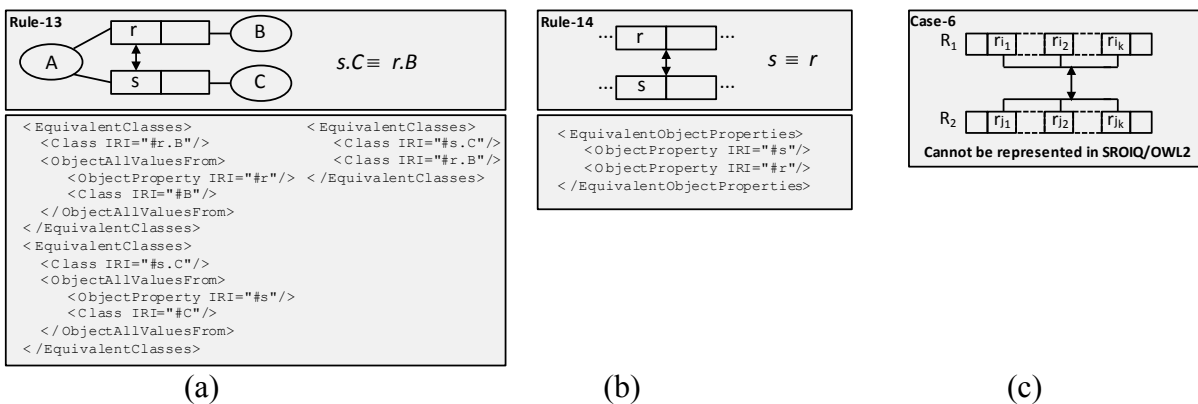

(a)   (b)   (c)

Figure 3.11: Equality Constraints

3.13 Exclusion Constraint

Similar to the subset and equality constraints, the exclusion constraint ($\otimes$) between roles and relations are mapped as shown in rules 15 and 16 (Fig.3.15) respectively. OWL 2 construct (DisjointObjectProperty) is a new feature of OWL 2 which states that two roles are disjoint to each other (It is representation in SROIQ is "Dis(s,r)").

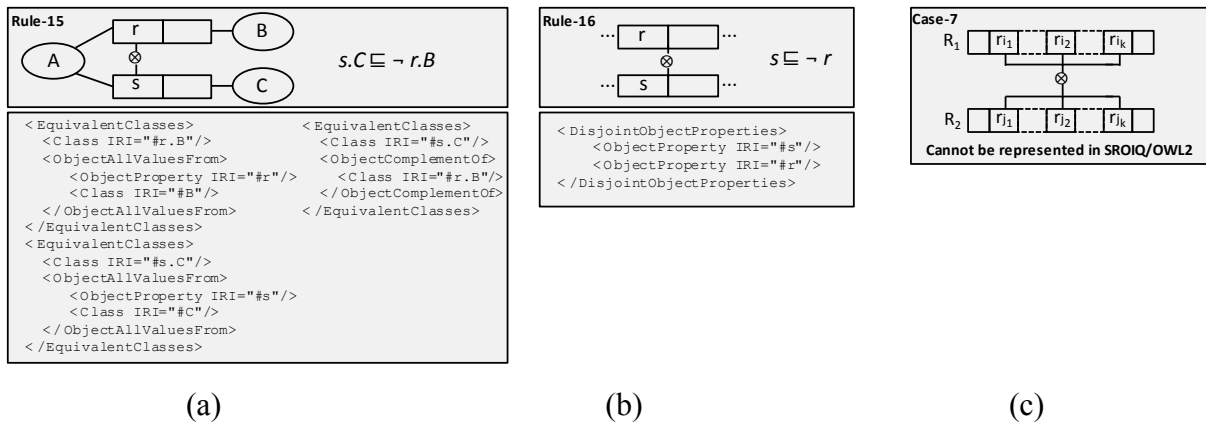(a)   (b)   (c)

Figure 3.12: Exclusion Constraints

3.14 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 super types. Six types of ring constraints are supported by ORM as illustrated in what follows.

OWL 2 supports Reflexive, Irreflexive, and Asymmetric object properties as a new feature in addition to Symmetric and Transitive that are supported by OWL 1.

3.14.1 Symmetric (sym):

This constraint states that if a relation holds in one direction, it also holds on the other. Fig. 3.13.a (Rule-17) depicts the general case formalization using SROIQ/OWL2.

3.14.2 Asymmetric (as):

Asymmetric constraint is the opposite of the symmetric constraint. If a relation holds in one direction, it cannot hold on the other. Fig. 3.13.b (Rule-18) depicts the general case formalization using SROIQ/OWL2.

3.14.3 Antisymmetric (ans):

The antisymmetric constraint is also an opposite of 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. Fig. 3.13.c (Case-8) shows an example of this constraint in addition to the general case formalization in SROIQ. Note that, up to our knowledge, this constraint cannot be expressed in OWL 2 because one cannot express role complement in OWL 2.

3.14.4 Irreflexive (ir)

The Irreflexive ring constraint states that an object cannot participate in a relation with himself. For example a person cannot be the 'parent of' himself (cannot play the role of 'ParentOf' with himself). However, for example, one can love himself, i.e., the '*love*' relation is reflexive; a ring constraint supported by SROIQ/OWL but not by ORM. Fig. 3.13.d (Rule-19) depicts the general case formalization using SROIQ/OWL 2.

3.14.5 Acyclic (ac):

The acyclic constraint is a special case of the irreflexive constraint. For example, stating that the relation 'ParentOf' is acyclic means that a person cannot be directly (or indirectly through a chain) 'ParentOf' himself. 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"[J07a].Up to our knowledge, acyclicity with any depth on binary relations cannot be represented in SROIQ/OWL 2 (see Fig. 3.13.e).

3.14.6 Intransitive (it):

A relation R is intransitive over its population $iff\ \forall x,y,z\ [R(x,y) \wedge R(y,z) \rightarrow \neg R(x,z)]$. For example, if a Person X is FatherOf Person Y and Y is FatherOf Z, then it cannot be that X is FatherOF Z. See Fig. 3.13.f for the general case formalization in SROIQ. However, as in the case of the antisymmetric constraint, this constraint cannot be expressed in OWL2 because one cannot express role complement in OWL2.
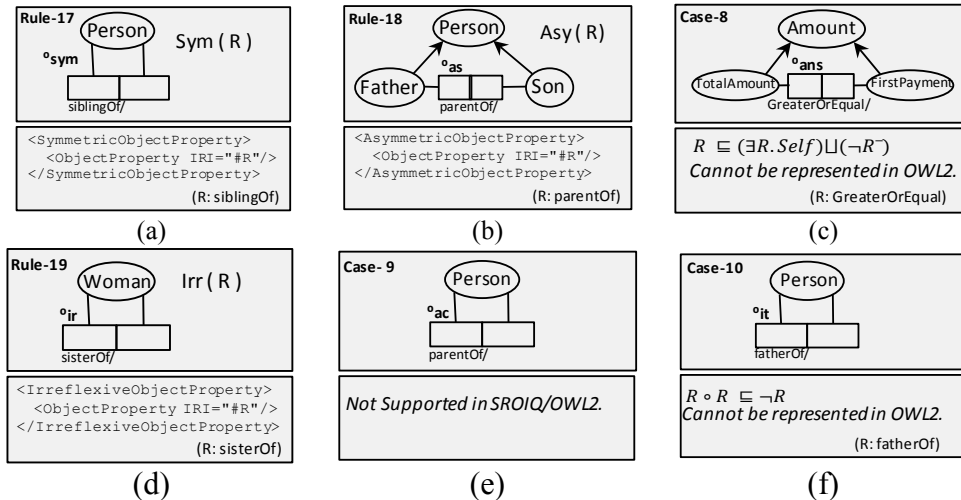
Rule-17 Person Sym (R)
ᵒsym
siblingOf/

```
<SymmetricObjectProperty>
  <ObjectProperty IRI="#R"/>
</SymmetricObjectProperty>
```
(R: siblingOf)

(a)

Rule-18 Person Asy (R)
Father ᵒas Son
parentOf/

```
<AsymmetricObjectProperty>
  <ObjectProperty IRI="#R"/>
</AsymmetricObjectProperty>
```
(R: parentOf)

(b)

Case-8 Amount
TotalAmount ᵒans FirstPayment
GreaterOrEqual/

$R \sqsubseteq (\exists R.Self) \sqcup (\neg R^{-})$
Cannot be represented in OWL2.
(R: GreaterOrEqual)

(c)

Rule-19 Woman Irr (R)
ᵒir
sisterOf/

```
<IrreflexiveObjectProperty>
  <ObjectProperty IRI="#R"/>
</IrreflexiveObjectProperty>
```
(R: sisterOf)

(d)

Case- 9 Person
ᵒac
parentOf/

Not Supported in SROIQ/OWL2.

(e)

Case-10 Person
ᵒit
fatherOf/

$R \circ R \sqsubseteq \neg R$
Cannot be represented in OWL2.
(R: fatherOf)

(f)

Figure 3.13: Ring Constraints

3.15 Objectified Relation

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 example in Fig. 3.14.a, each (*Student*, *Vehicle*) enrollment is treated as an object-type that scores a *Grade*. 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. Objectified relations cannot be represented in SROIQ/OWL 2 as the additional uniqueness axiom cannot be represented in SROIQ/OWL 2. Refer to [J07a] for the representation of objectified relations in DLR description logic.

3.16 Syntatic Sugar for ORM/OWL 2

ORM and OWL 2 provide syntactic sugar formalization to ease the modeling of some constraints, such as identity, total and exclusive constraints which are illustrated below.

3.16.1 Identity Constraint:

This constraint determines the unique identifier of an object-type, and is usually specified inside the ellipse of the object-type (see Fig. 3.14.b). This constraint implies two constraints: role mandatory (section 4.5.1) and role uniqueness (section 4.6.1). This means that the object-type, specified as 'key' in Fig. 3.14.b, uniquely identifies the object-type A: i.e., it is

mandatorily related to the object-type A and each instance of 'key' is related with at most one instance of A and each instance of A is related to at most one instance of 'key' . Although this constraint is not supported by SROIQ, it is mapped to OWL2 using the '*HasKey*' construct.

3.16.2 Total and Exclusive Constraints:

The Total and Exclusive constraints (section 3.5 and 3.6) often appear together in ORM. Instead of using two separate OWL2 constructs for their mappings (*ObjectUnionOf* and *DisjointClasses*), OWL2 provides one construct, namely, *DisjointUnion*, which fulfills both Total and Exclusive Constraints (Fig. 3.14.c).

(a)   (b)   (c)
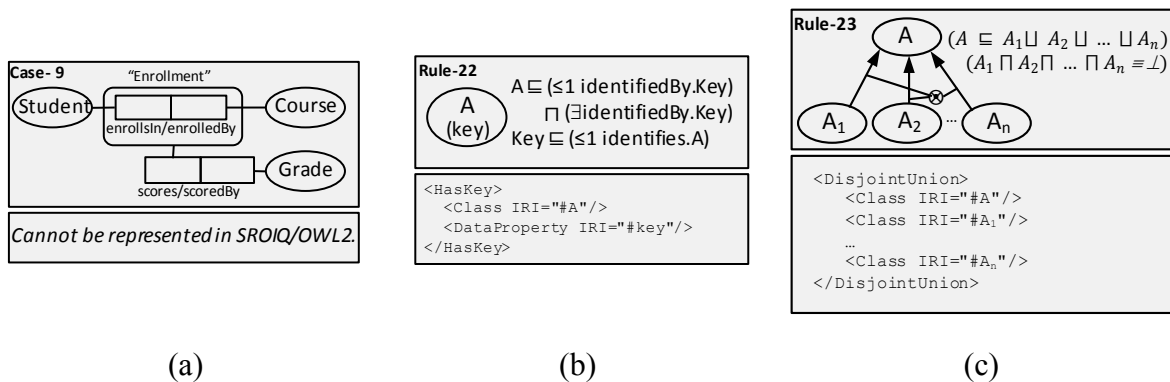
Figure 3.14: (a) Objectified Relations, (b) Identity Constraint, (c) Total and Exclusive Constraints

Chapter Four

Extending ORM for Complete representation of OWL 2

4.1 Introduction

In chapter 3 of this thesis, we mapped all ORM constructs to SROIQ/OWL2. This allows one to build his/her ontology graphically using ORM and then map it automatically into OWL2. However, the current graphical notations of ORM are not sufficient to represent all OWL2 constructs (i.e., some OWL2 constructs cannot be represented graphically using ORM). In this section, we extend the ORM graphical notation to represent the eleven OWL2 constructs/expressions not covered currently by ORM. These OWL2 constructs are: (i) Equivalent Classes, (ii) Disjoint Classes, (iii) Intersection of Class Expressions, (iv) Class Complement, (v) Class Assertions, (vi) Individual Equality, (vii) Individual Inequality, (viii) Positive Object/Data Property Assertions, (ix) Negative Object/Data Property Assertions, (x) Reflexive, (xi) and Transitive. In addition we use already existing notations to represent Thing and Nothing Classes , and also Top and Bottom Object/Data Properties. Some of the OWL 2 expressions cannot be logically represented in a graphical format, so we use non-graphical notations for representing these OWL 2 expressions to completely cover OWL 2. The OWL 2 expressions that are not represented graphically and represented in non-notational expressions include (i) Datatypes, Facets, and Data Range Expressions, and (ii) Annotations. The graphical notations that we developed for representing the above mentioned eleven constructs in ORM were evaluated by means of a survey that included 31 practitioners in the field of ORM and OWL. After the evaluation process, final notations were chosen based on the results of the

survey. In this way of complete representation of OWL 2 using ORM we can build an ontology graphically using ORM under the semantic of OWL 2. Using this mechanism we can combine both the simplicity of representing an ontology and the full functionality needed to check the correctness of the built ontology. The resulting OWL 2 ontology is supported by many sound and complete reasoners that can verify its correctness. Section 4.2 – 4.8 briefly discusses those eleven OWL2 constructs and our proposed graphical notations for their representation in ORM.

4.2 Equivalent Classes

An Equivalent Class constraint in SROIQ/OWL 2 states that all classes constrained by this rule are semantically equivalent to each other. This rule is expressed in OWL2 using the **EquivalentClasses** construct and in SROIQ description logic using the notation of '$\equiv$'. However, no notation is available in ORM for representing this construct, because ORM proposes that there is no need for equivalent objects within the same modeling case. However, because of the rapidly increasing usage of ontologies in data integration, the Equivalent Classes constraint is highly needed. The proposed notation for representing this constraint using ORM is shown in Fig. 4.1 along with a clarifying example. It's worth mentioning that this graphical notation was preferred by a (50%) of practitioners who participated in the evaluation survey.
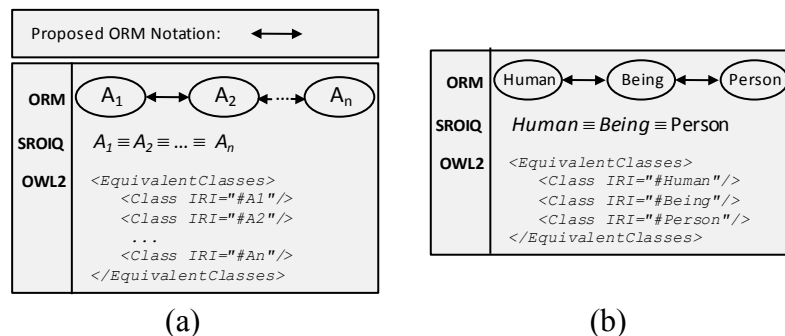
Figure 4.1: Equivalent Classes Constraint

4.3 Disjoint Classes

The population of all classes constrained by the Disjoint Classes constraint is pairwise distinct, i.e., the intersection of the population of each pair of the constrained classes must be empty. This rule exists in ORM. However, it is restricted to be used between subtypes that belong to

the same supertype. As, in ORM, all objects within a modeling case are assumed to be disjoint with each other. This rule is expressed in OWL2 using the **DisjointClasses** construct. For representing this constraint in ORM, we propose to use the notation of ($\otimes$), as shown in Fig. 4.2. It's worth mentioning that, from the three suggested graphical notations in the survey, this graphical notation was chosen by (64%) of practitioners who participated in the evaluation survey.
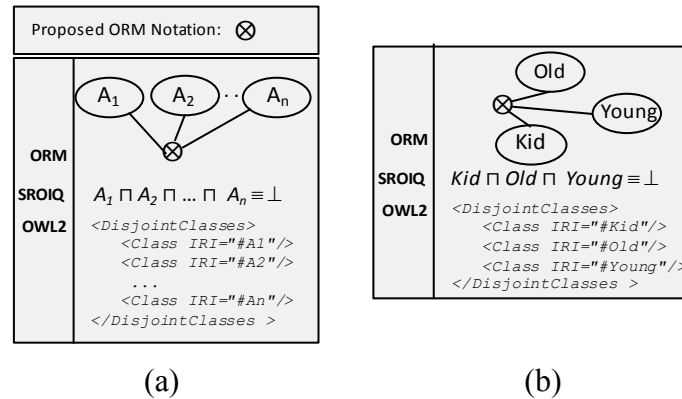

(a)   (b)

Figure 4.2: Disjoint Classes Constraint

4.4 Intersection of Class Expressions

The intersection of classes A and B is all the individuals (instances) of A that are also instances of B but no other instances. This expression cannot be expressed currently in ORM. In OWL2, **ObjectIntersectionOf** expression is used to represent the intersection of classes, whereas the notation of '$\sqcap$' is used for the SROIQ representation, as shown in Fig. 19. In ORM, we propose to use the notation of '$\sqcap$' inside a bubble located on the edge of an ellipse (see Fig. 4.3). This bubble connects directly via lines to the classes to be intersected. Inside the ellipse, the name of the class equivalent to the intersection expression is assigned. Note that the intersection bubble can be connected directly to many classes: the classes which are being intersected. No other relations are allowed to be connected through the bubble. However, the ellipse (which represents the equivalent class of the intersection expression) can be connected via any relation to any other class.
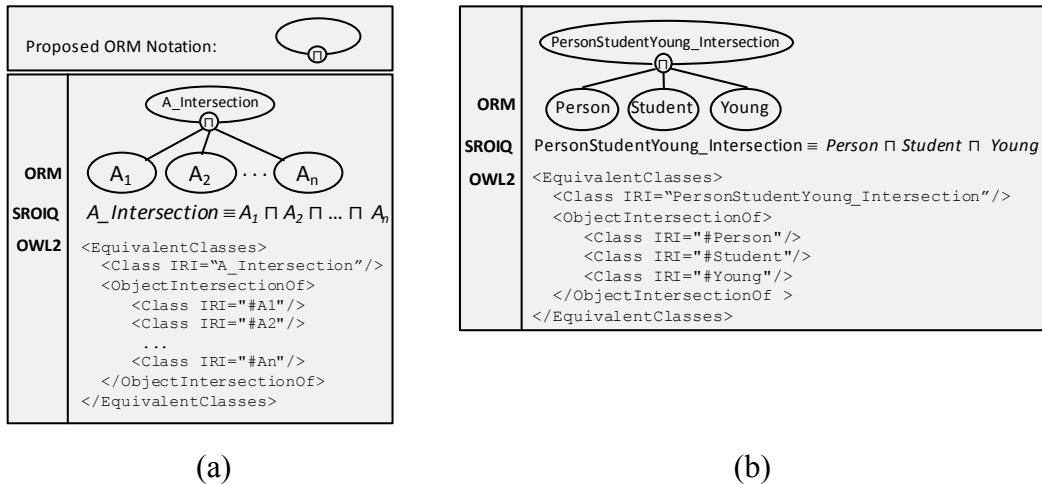
Figure 4.3: Intersection of Class Expressions

4.5 Class Complement

The complement of class *A* refers to the population of the Universe of Discourse (UoD) that is not of A (i.e., the instances outside of A). This expression cannot be expressed currently in ORM. In OWL2, **ObjectComplementOf** expression is used to represent class complement, whereas the notation of '¬' is used for the SROIQ representation, as shown in Fig. 4.4. In ORM, similar to the Intersection of Class Expressions discussed above, we propose to use the notation of '¬' inside a bubble located at the edge of an ellipse (see Fig. 4.4). This bubble connects directly via a line to the class to be complemented. Inside the ellipse, the name of the class equivalent to the complement expression is assigned. Note that the complement bubble can only be connected directly to one class: the class which is being complemented. No other relations are allowed to be connected through the bubble. However, the ellipse (which represents the equivalent class of the complement expression) can be connected via any relation to any other class.
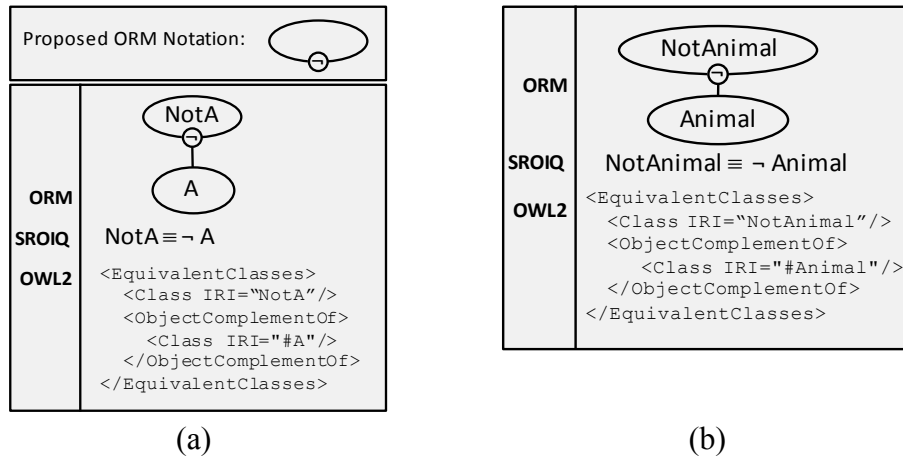
Figure 4.4: Class Complement

4.6 Universal and Empty Classes

Two classes in OWL 2 are predefined, namely, the classes "owl:Thing" and "owl:Nothing". "owl:Thing" is referred to as the Universal Class while "owl:Nothing" is called the Empty Class. The extension of class "owl:Thing" (i.e., its instances) is the set of all instances in the Universe of Discourse (UoD). Thus, all classes are subclasses (i.e., subtypes) of this universal class. On the other hand, the extension of class "owl:Nothing" is the empty set. Consequently, the empty class is a subclass of all classes. In SROIQ, the universal and empty classes correspond to the Top ($\top$) and Bottom ($\bot$) concepts, respectively. These two predefined OWL 2 classes are not currently defined in the ORM notation. We propose to express these two classes using the regular ORM object-type notation as show in Fig. 4.5. Note that this proposed graphical representation was not evaluated in the survey because of its intuitiveness. That is, the representation of these two predefined OWL2 classes is no different than the representation of any other OWL2 class; all OWL2 classes are mapped in ORM as object-types.
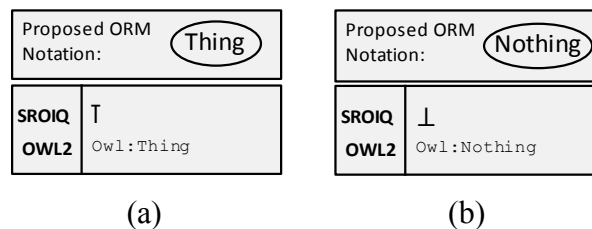
Figure 4.5: Thing and Nothing Classes

4.7 Universal and Empty Object/Data Properties

OWL 2 provides two built-in object/data properties with predefined semantics: (i) `owl:topObjectProperty`/ `owl:topDataProperty` (Universal Property), (ii) `owl:bottomObjectProperty`/ `owl:bottomDataProperty` (Empty Property). The universal object property connects all possible pairs of object-type instances (individuals) while the universal data property connects all possible object-type instances (individuals) with all values (literals). On the contrary, the empty property neither connects any pair of object-type instances (individuals) nor connects any object-type instance (individual) with a value (literal). Unlike other variants of Description Logic, SROIQ does support Universal and Empty roles. These predefined OWL 2 properties are not currently defined in ORM. We propose to express these notations using the regular ORM role notation as show in Fig. 4.6. Note that this proposed graphical representation was not evaluated in the survey because of its intuitiveness. That is, the representation of these predefined OWL2 properties is no different than the representation of any other OWL2 property; all OWL2 properties are mapped as ORM roles.
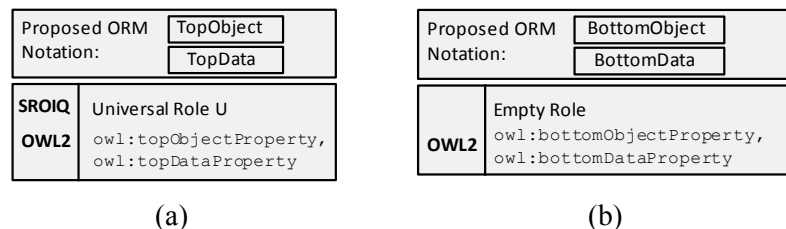
| Proposed ORM Notation: | TopObject |
| | TopData |

| SROIQ OWL2 | Universal Role U `owl:topObjectProperty,` `owl:topDataProperty` |

| Proposed ORM Notation: | BottomObject |
| | BottomData |

| OWL2 | Empty Role `owl:bottomObjectProperty,` `owl:bottomDataProperty` |

(a)   (b)

Figure 4.6: Top and Bottom Object/Data Properties

4.8 Class Assertions

The **ClassAssertion** axiom of OWL2 allows one to state that an individual is an instance of a particular class [MPP09]. In SROIQ, this is done in the assertion component, i.e., the ABox which contains instantiations of the axioms specified in the TBox. In ORM, we propose to use the notation depicted in Fig. 4.7. This notation provides the user the flexibility to show/hide the instances of a particular class. In our implementation of this notation in DogmaModeler (discussed in chapter 6), clicking on the (▶) symbol at the bottom of the ellipse expands/collapses the set of instances. The user specifies how many instances he/she prefers to be shown.
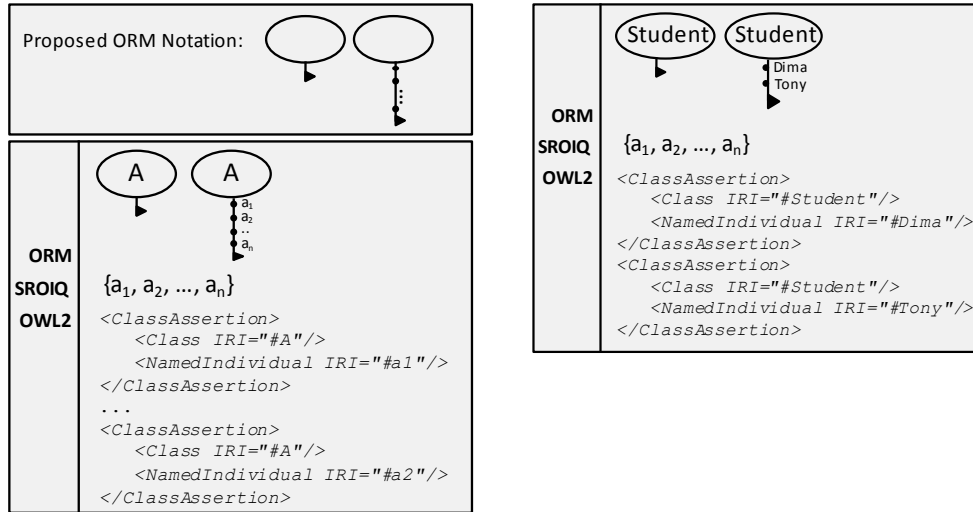
(a)   (b)

Figure 4.7: Class Assertions

4.9 Individual Equality

The OWL2 individual equality axiom **SameIndividual** states that all of the individuals constrained by this rule are equal to each other. In SROIQ, this axiom is expressed in the assertion component, i.e., the ABox, using the notation of '=' between individuals. In ORM, we propose to use the notation of ($\ominus$) to express individual equality. This notation is used between class instances as shown in Fig. 4.8.
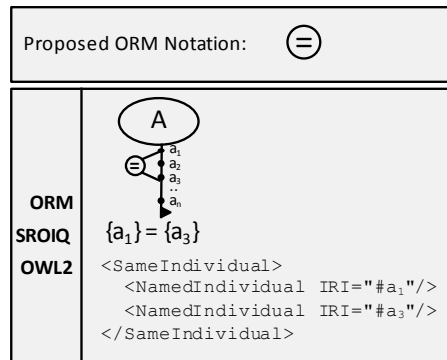
Figure 4.8: Individual Equality

4.10 Individual Inequality

The OWL 2 individual inequality axiom **DifferentIndividuals** states that all of the individuals constrained by this rule are different from each other. In SROIQ, this axiom is expressed in the assertion component, i.e., the ABox, using the notation of '$\neq$' between individuals. In ORM,

we propose to use the notation of ($\neq$) to express individual inequality. This notation is used between class instances as shown in Fig. 4.9.
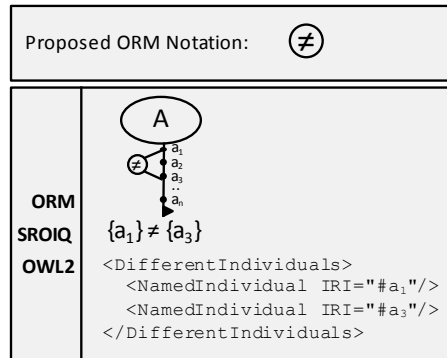
Figure 4.9: Individual Inequality

4.11 Property Assertions

4.11.1 Positive Object/Data Property Assertion

The OWL2 **Object/DataPropertyAssertion** states that the individuals or individuals and values constrained by this rule are related to each other by the specified object or data property. In SROIQ, this axiom is expressed in the assertion component, i.e., the RBox, using the notation of '+' between individuals or individuals and values. In ORM, we propose to use the notation of (+) to express the relation between individuals. This notation is used between class instances that are related with each others as shown in Fig. 4.10.
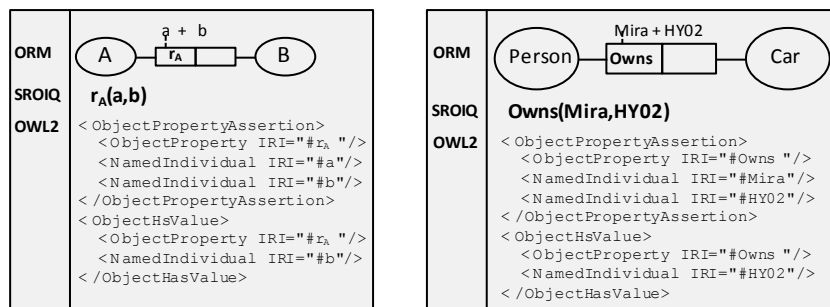

    (a)   (b)

Figure 4.10: Positive Object Property Assertion

4.11.2 Negative Object/Data Property Assertion

The OWL2 **NegativeObject/DataPropertyAssertion** states that the individuals or individuals and values constrained by this rule are *not* related to each other by the specified object or data

property. In SROIQ, this axiom is expressed in the assertion component, i.e., the RBox, using the notation of '-' between individuals or individuals and values. In ORM, we propose to use the notation of (-) to express the relation between individuals. This notation is used between class instances that are *not* related with each others as shown in Fig. 4.11.
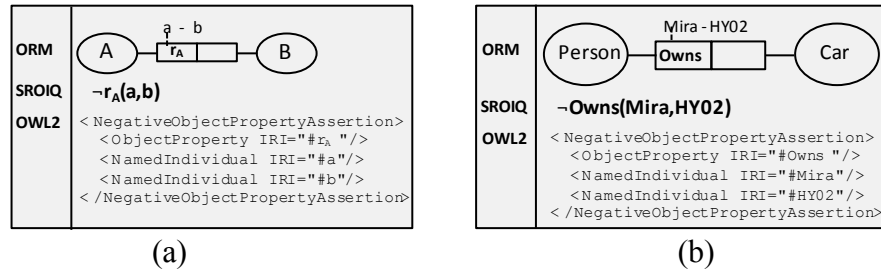
Figure 4.11: Negative Object Property Assertion

4.12 Reflexive (ref)

This constraint states that an object can participate in a relation with himself. For example, a 'person' can love himself (he can play the role of 'loves' with himself). Reflexive is not used in ORM, Where it can be an extension to ORM notations as we want to reflect the semantic of OWL 2 into ORM. Fig. 4.12.a depicts the general case extension of this constraint using ORM, where this notation is selected according to similar notations of ring constraints shown in Fig. 3.16.

4.13 Transitive (tra)

A relation R is transitive over its population $iff\ \forall x, y, z\ [R(x,y) \wedge R(y,z) \rightarrow R(x,z)]$. For example, if a Person X is FrindOf Person Y and Y is FriendOf Z, then X is FriendOf Z. Transitive does not exist in ORM, Where it can be an extension to ORM notations as we want to reflect the semantic of OWL 2 into ORM. Fig. 4.12.b depicts the general case extension of this constraint using ORM, where this notation is selected according to similar notations of ring constraints shown in Fig. 3.16.
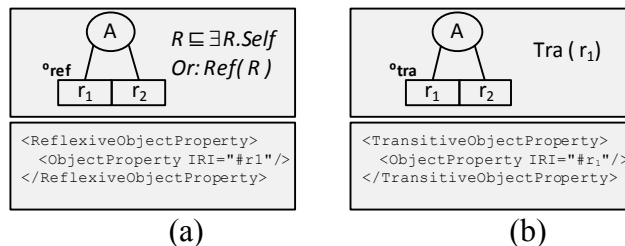
Figure 4.12: Extended Ring Constraints

4.14 Non Notational Expressions

In the previous subsections, we have introduced our proposed *graphical* extension of ORM. However, this extension does not cover all OWL 2 expressions graphically. The OWL 2 expressions not expressed graphically can be put into two categories; (i) Datatypes, Facets, and Data Range Expressions, and (ii) Annotations.

OWL 2 introduces many built in datatypes in addition to the "Number" and "String" datatypes defined in ORM, such as Real, Rational, Double, Float, Boolean, Binary, etc. In addition, it introduces the so-called "Facets", borrowed from XML Schema Datatypes, which are simply a group of restrictions used to specify new user-defined datatypes. For example, one can define a new datatype for a person's age, called *personage*, by constraining the datatype *Integer* to values between 0 and 150 (inclusive) using the *minInclusive* facet. Furthermore, OWL 2 supports advanced uses of datatypes, called Data Range Expressions, which include expressions similar to those used with OWL 2 Classes such as complement, union and intersection. For example, assuming we have already defined a datatype called *minorAge*, we can define the datatype *majorAge* by complementing the datatype *minorAge* and then intersecting it with the datatype *personAge*. All OWL 2 Datatypes, Facets, and Data Range Expressions are not expressed graphically. Instead, we introduce the "Datatypes Guided Editor" which aids the user in specifying datatypes and defining new datatypes and data range expressions. Fig. 4.13.a depicts a simplified Datatypes Guided Editor as implemented in our DogmaModeler tool.

The last category of our non-notational expressions is OWL2's "Annotations". In OWL 2, annotations are used to describe parts of the OWL 2 ontology or the ontology itself. For example, an annotation can be simply a human-readable comment on an axiom of an ontology. E.g., one can add the following comment on the subtype relation between Man and Person: "This subtype relation states that every man is a person". Other annotation properties include: Label, SeeAlso, VersionInfo, etc. Annotations can be specified using the "Annotations Guided Editor". Fig. 4.13.b depicts a snapshot of our implemented annotations editor part of DogmaModeler.
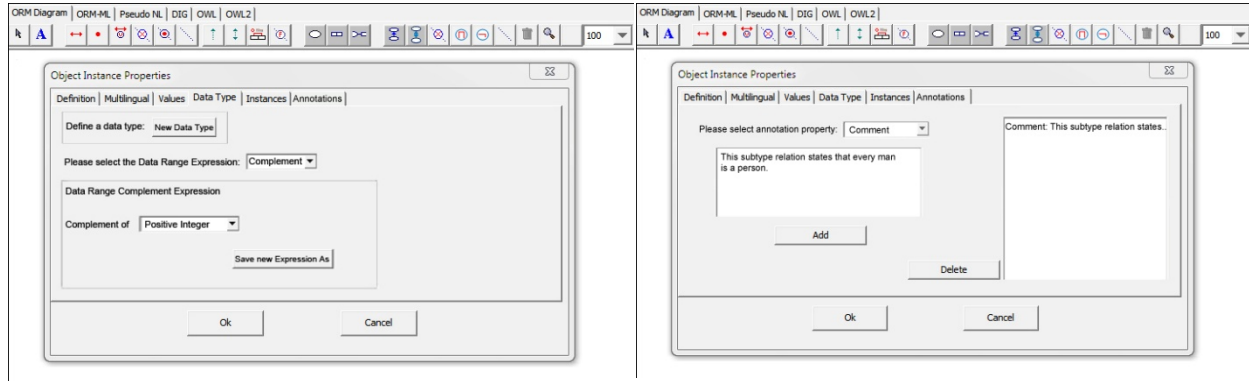
(a)   (b)

Figure 4.13: The Datatypes and Annotations Guided Editors of DogmaModeler

An illustration of full representation for mapping OWL 2 into ORM is shown in Appendix B.

4.15 Use Case

One of the most important use cases of the extended ORM notation introduced above is the case of integration. Consider, for example, the case of ontology integration in Fig. 4.14. The figure depicts two sample ontologies expressed in ORM; the first ontology (Ontology-1) represents a part of the organization (non-natural person) ontology specifying, in particular, the *Company* and the *Local Government Unit* entities. The second ontology represents another part of the organization ontology, namely, the *Association* entity. Note that these two sample ontologies are based on the Palestinian Legal Person Ontology [DJF11] and thus are in harmony with the Palestinian law. However, what is depicted in Figure 4.14 is not a complete ontology and is only meant to be a sample demonstration of the usefulness of the newly proposed ORM notations.
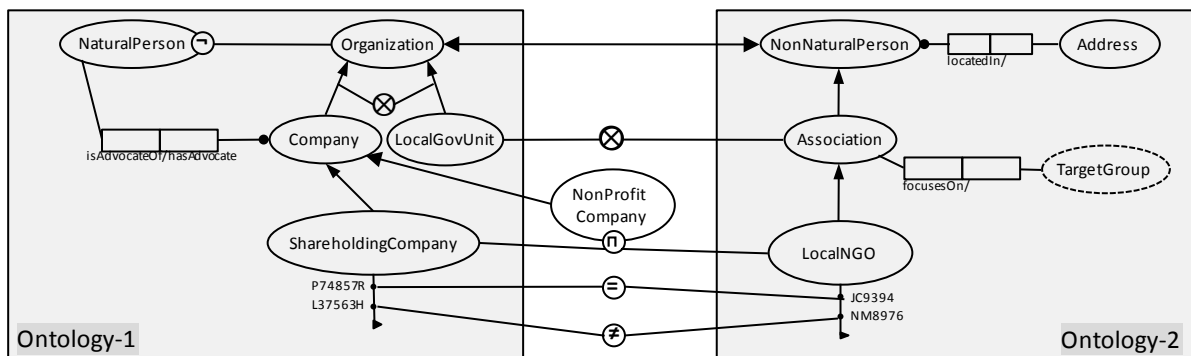
Figure 4.14: Use Case of the extended ORM notations.

Let us look at the usage of the new ORM notations in Ontology-1. In this ontology, the new notation is used in two places; (i) to express the class 'Natural Person' as the complement of the 'Organization' entity and (ii) to express two assertions of the 'Shareholding Company' entity. In Ontology-2, we also use class assertions to represent two 'LocalNGO' instances. For the purpose of integrating the two ontologies the following newly introduced notations are used:

i) *Equivalent Classes,* expressed as a double-headed arrow between 'Organization' and 'Non-Natural Person' entities, to express the fact that both entities are equivalent.

ii) *Disjoint Classes,* expressed using the notation of "⊗" between 'LocalGovUnit' and 'Association' to express the fact that both entities are disjoint.

iii) *Intersection of Class Expressions,* used to introduce a new entity, namely, the 'Non Profit Company' which includes all shareholding companies that are also registered as local NGOs. Note that, this type of companies does exist in Palestine; it includes private limited-liability companies that provide services to the society but whose profit is not allowed to be distributed among the partners.

iv) *Individual Equality*, expressed using the notation of (⊜) between the shareholding company (P74857R) and the Local NGO (JC9394). This means that the two identifiers refer to the same real-world entity. Notice the usage of the Individual Equality here for Entity Resolution/ Disambiguation: a much-used process to match different identifiers from different heterogeneous information systems that refer to the same entity.

v) *Individual Inequality,* expressed using the notation of (≢) and is used to state that the two instances constrained by this rule are not the same. In our example, this expression is also used for entity disambiguation by stating that that shareholding company (L37563H) and the local NGO (NM8976) refer to different real-world organizations.

The complete mapping of the ORM diagram in Fig. 4.14 into OWL2 is provided in Appendix A-2.

Chapter Five

Evaluation

5.1 Introduction

Evaluation is divided into two parts. For the first part of evaluation we discuss the evaluation of our work of mapping/formalizing ORM into OWL 2/SROIQ. In particular, we briefly discuss the means by which we evaluated our work and give some clarifying examples. For this evaluation, the last version (version 2) of RacerPro 2[14] was used as a description logic reasoning tool, especially because of its support of OWL 2. RacerPro provides an interface to query and reason about knowledge bases. A knowledge base in RacerPro consists of a T-Box and an A-Box. For each of the formalized ORM notations, its OWL 2 mapping was inserted into the RacerPro system in the T-Box of a Knowledge Base. After that, the knowledge base was populated with several A-Box assertions in ordered to perform various kinds of tests and queries over it.

For the second part, we evaluate our proposed ORM extension via a survey that was made to choose the most appropriate graphical notations. This survey involved practitioners in the fields of ORM and OWL 2.

5.2 Evaluation Process for mapping ORM into OWL 2

For each mapping work, tests were done to be sure of the correctness of the mapped OWL 2 construct. These tests include consistency, coherency, and instance checks in addition to

---

[14] http://www.racersystems.com/products/racerpro/users-guide-1-9-2-beta.pdf

several other checks depending on the individual construct to be tested. The consistency test checks whether the given Assertion Box is consistent with respect to the Terminology Box. An A-Box *A* is consistent with a T-Box *T iff* A-Box has a model *w.r.t.* T-Box. Otherwise, the A-Box is called inconsistent. Coherency (also called Satisfiability) can be divided into three tests: Concept Coherency, T-Box Coherency, and Ontology Coherency. Checking the satisfiability (coherency) of a concept *C* in a T-Box *T* means to check whether there exists a model *I* of *T* such that $C^I \neq \phi$. This is usually done by checking this concept for any possible contradictions in the T-Box. If the concept *C* is involved in a contradiction, this means that it cannot be satisfied. For the T-Box coherency; a T-Box *T* is said to be incoherent *iff* there exists an unsatisfiable concept in *T*. Similarly, an ontology *O* is incoherent *iff* its T-Box is incoherent. The Instance Retrieval test simply finds all individuals (instances) mentioned in an A-Box that are instances of a certain concept *C*. Different kinds of queries supported by the new query language (nRQL) of RacerPro 2.0 were used to retrieve the instances of classes within the needed ontology (instance checking) to check the correctness of the ontology and consequently the mapping itself. It is worth noting here that all A-Box tests and queries are written in nRQL (The New RacerPro Query Language), a query language for the RacerPro system that is capable of querying description logics, RDF(s), and OWL.

All possible cases of instance checking are taken into consideration to evaluate mapping correctness. In what follows, we illustrate the evaluation of mapping ORM into OWL 2 for each mapped construct.

5.2.1 Binary Relation (Rule 1):

A general case of Binary Relation is plotted and mapped into OWL 2 functional syntax (Fig. 5.1) consisting of two classes (objects) Person and Vehicle, and two relations Drives and DrivenBy. The instances that are inserted for classes Person and Vehicle are shown in Fig. 5.1. As an evaluation process, all possible cases of tests are taken into consideration to prove the correctness of the mapping as follows:

i)   The domain of relation Drives which is expected to be class Person is checked and the right expected answer (Person) is obtained.

ii) The range of Drives which is expected to be class Vehicle is checked and the right expected answer (Vehicle) is obtained.

iii) Using the same way, the domain and range for relation DrivenBy which are expected to be Vehicle and Person respectively are checked and the right expected answers (Vehicle and Person) are obtained.

iv) The inverse of role Drives which is expected to be DrivenBy is checked and the right expected answer (DrivenBy) is obtained.

v) Instances of classes Person and Vehicle are retrieved and the right expected instances are obtained.

vi) Each instance of class Person which is related to the other instance of class Vehicle by the role Drives is retrieved and the right expected answer is obtained. The same is done for the role DrivenBy and the right expected answer is obtained. That was done to make sure that Drives is the inverse role of DrivenBy and vice versa.

All the preceding mentioned cases are shown in Figure 5.1.
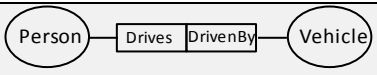
OWL 2 Functional Syntax Declaration(NamedIndividual(:p11)) ClassAssertion(:Person :p11) Declaration(NamedIndividual(:p12)) ClassAssertion(:Person :p12) Declaration(NamedIndividual(:p13)) ClassAssertion(:Person :p13) Declaration(NamedIndividual(:v11)) ClassAssertion(:Vehicle :v11) Declaration(NamedIndividual(:v12)) ClassAssertion(:Vehicle :v12) Declaration(NamedIndividual(:v13)) ClassAssertion(:Vehicle :v13)	Checked Example	Person — Drives DrivenBy — Vehicle	
	Type of Test	**nRQL query**	**Result**
	Domain	(role-domain |#Drives|)	:Person
		(role-domain |#DrivenBy|)	:Vehicle
	Range	(role-range |#Drives|)	:Vehicle
		(role-range |#DrivenBy|)	:Person
	Inverse	(role-inverse |#Drives|)	:DrivenBy
	Retrieve Person and Vehicle Instances	(retrieve(?x)(?x #Person))	:p13, :p12, :p11
		(retrieve(?x)(?x #Vehicle))	:v13, :v12,:v11
	Retrieve related Instances	(retrieve (?x ?y) (?x ?y #Drives))	(:p11, :v11),(:p12, :v12),(:p13, :v13)
		(retrieve(?x?y)(?x?y #DrivenBy))	(:v11, :p11),(:v12, :p12), (:v13, :p13)

Figure 5.1: Tests performed on OWL 2 mapping of ORM binary role notation (Rule 1)

5.2.2 Unary Relation (Rule 2):

An example (Figure 5.2) in OWL 2 is created to formalize the Unary Relation of ORM. The example contains a class named Person and a data property named smokes with Boolean

range, then we asserted two instances for class Person related by data property smokes where one of these instances set to be true by assigning "1" to it and the other set to false by assigning "0" to it . When we retrieve the instances (Table 5.1) of property smokes, it gives us the population of class person which plays data property range of type Boolean.
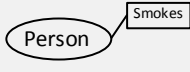
OWL 2 Functional Syntax	Checked Example	Person Smokes	
Declaration(Class(:Person)) Declaration(DataProperty(:smokes)) SubDataPropertyOf(:smokes owl:topDataProperty) DataPropertyRange(:smokes xsd:boolean) Declaration(NamedIndividual(:ahmad)) ClassAssertion(:Person :ahmad) DataPropertyAssertion(:smokes :ahmad "1"^^xsd:boolean) Declaration(NamedIndividual(:sahar)) ClassAssertion(:Person :sahar) DataPropertyAssertion(:smokes :sahar "0"^^xsd:boolean)	**Type of Test**	**nRQL query**	**Result**
	Retrieve Instances	(retrieve(?x(datatype-fillers(|#smokes|?x)))(?x (some |#smokes|)))	(((:ahmad) ((:datatype-fillers (:smokes)) (#T)))((((:sahar) ((:datatype-fillers (:smokes))(#T))))
DifferentIndividuals(:sahar :ahmad) DataPropertyAssertion(:smokes :ahmad "1"^^xsd:boolean) DataPropertyAssertion(:smokes :sahar "0"^^xsd:boolean)	Consistency Check	(abox-consistent? file://unaryrelation.owl)	True

Figure 5.2: Tests performed on OWL 2 mapping of ORM unary role notation (Rule 2)

5.2.3 Subtype (Rule 3):

The OWL 2 file which is mapped from ORM is loaded into RacerPro 2.0. The file includes two classes Man and Person (where Man is a SubClass of Person) is created (Figure 5.3). Instances for both classes are inserted. Two types of tests were performed, namely, concept subsumption and instance retrieval. In short, the tests we have done on the subtype relation can be summarized as follows. After inserting the OWL 2 mapping of the ORM in Fig. 5.3 as a T-Box in a knowledge base, we entered a set of assertions into the A-Box. These assertions were: one assertion of the object-type *Person* (i.e., Mira) and two assertions of the object-type *Male* (Issa and Abbas). Two concept consumption tests were performed; the first to check whether the object-type *Male* subsumes *Person*, resulting in 'True', as expected. The second verified whether *Person* subsumes *Male*, resulting in Nil (False), also as expected. When we retrieved the instances of *Person*, all instances of *Male* (Issa and Ahmad) appeared in the result set in addition to the instance of *Person* (Mira); a result which we expected because of the subtype relation.

All possible cases for ontology checking are taken into consideration to evaluate mapping correctness as follows:

i) Concept subsumption is checked where class Person subsumes class Man and the expected result (which is true) is obtained. The inverse is done where does class Male subsumes class Person and the expected result (which is false) is obtained.

ii) The instances of class Person are retrieved and the expected results are the instances already asserted for class Person (super class) in addition to the instances of class Male (subclass). These results are obtained.

iii) The instances of class Male are retrieved and those are expected to be only the instances assigned to class Male and this result is obtained.
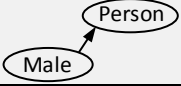
OWL 2 Functional Syntax	Checked Example		
SubClassOf(:Male :Person) Declaration(NamedIndividual(:Issa)) Declaration(NamedIndividual(:ahmad)) Declaration(NamedIndividual(:Mira)) ClassAssertion(:Male :Issa) ClassAssertion(:Male :Ahmad) ClassAssertion(:Person :Mira)	**Type of Test**	**nRQL query**	**Result**
	Concept Subsumption	?(concept-subsumes? |#Person| | #Male|)	True
		?(concept-subsumes?|#Male| | #Person|)	Nil (i.e., False)
	Retrieve Instances	?(retrieve(?x) (?x #Person))	Mira, Issa, Ahmad
		?(retrieve(?x) (?x #Male))	Issa, Ahmad

Figure 5.3: Tests performed on OWL 2 mapping of ORM subtype role notation (Rule 3)

5.2.4 Total Constraint (Rule 4):

An example (Fig. 5.4) that contains class Person which is equivalent to class Man or Female is created. Instances for the three classes are asserted.

For Total Constraint, all possible cases (Fig. 5.4) are taken into consideration as follows:

i) Checking if class Person (super class) is equivalent to the union of classes Male and Female, and the expected result which is true is obtained.

ii) The instances of class Person are retrieved and those are expected to be the already assigned instances for Person in addition to instances of class Male or Female. The expected result is obtained.

iii) The instances of subclasses Male and Female are expected to be just the instances assigned to each class and the result is as expected.
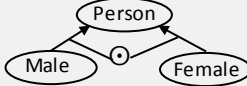
OWL 2 Functional Syntax	Checked Example	Person / Male ⊙ Female		
Declaration(Class(:Male))				
Declaration(Class(:Person))	**Type of Test**	**nRQL query**		**Result**
EquivalentClasses(:Person ObjectUnionOf(:Female :Male))	Concept-Equivalence	(concept-equivalent? |#Person|(or |#Male| |#Female|))		True
Declaration(Class(:Female)) Declaration(NamedIndividual(:ahmad)) ClassAssertion(:Male :ahmad)	Retrieve Instances	(retrieve(?x)(?x #Male))		((((:ahmad)))
Declaration(NamedIndividual(:issa)) ClassAssertion(:Person :issa)		(retrieve(?x)(?x #Female))		((((:wafa)))
Declaration(NamedIndividual(:wafa)) ClassAssertion(:Female :wafa)		(retrieve(?x)(?x #Person))		(((:issa))((:ahmad))((:wafa)))

Figure 5.4: Tests performed on OWL 2 mapping of ORM total constraint (Rule 4)

5.2.5 Exclusive Constraint (Rule 5):

An example (Figure 5.5) that contains classes Male and Female which are disjointed is created. The same instance is created for both classes. For Exclusive Constraint, the cases (Fig. 5.5) are :

i) Disjoint is checked between classes Male and Female and the result is true as expected true.

ii) The equivalence of class Person to the union of classes Male and Female is checked and the result is false as expected.

iii) The instance ahmad (as an example) is assigned to both classes Male and Female, where in this assignment the Abox consistency is violated to ensure that classes Male and Female are disjoint. The result is as expected inconsistent Abox.
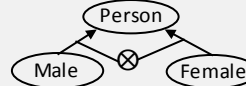
OWL 2 Functional Syntax	Checked Example	Person / Male ⊗ Female	
Declaration(Class(:Male)) Declaration(Class(:Female))			
Declaration(Class(:Person)) SubClassOf(:Male :Person)	**Type of Test**	**nRQL query**	**Result**
SubClassOf(:Female :Person)	Concept-Disjoint	?(concept-disjoint? |#Male| |#Female|)	True
DisjointClasses(:Male :Female) Declaration(NamedIndividual(:ahmad))	Concept-Equivalence	?(concept-equivalent? |#Person|(or |#Male| |#Female|))	Nil
ClassAssertion(:Male :ahmad) ClassAssertion(:Female :ahmad)	Consistency Check	? (abox-consistent? file:// /disjointclasses.owl) (with assertions that <u>do not</u> violate the constraint)	True
Declaration(NamedIndividual(:wafa)) ClassAssertion(:Female :wafa)		? (abox-consistent? file://disjointclasses.owl) (with assertions that <u>do</u> violate the constraint)	Nil

Figure 5.5: Tests performed on OWL 2 mapping of ORM exclusive constraint (Rule 5)

5.2.6 Role Mandatory (Rule 6):

To represent the ORM Role Mandatory constraint in OWL 2, we use the construct "ObjectSomeValuesFrom" that represents the extensional quantifier in SROIQ. Example of section 5.2.8 (Figure 5.7) includes the representation of Role Mandatory constraint. The relation "OwnedBy" is populated by relating the instances of class "Company" to that of class "Vehicle". When checking the consistency of ABox, the result was that the ABox is consistent as the constraint Mandatory (represented by "ObjectSomeValuesFrom") is achieved.

5.2.7 Disjunctive Mandatory (Rule 7):

An example (Figure 5.6) that contains the classes and relations of Rule 7. Instances (created to prove the correctness of formalization) are asserted and related by OwnedByC and OwnedByP roles. When we check the coherency of TBox and consistency of ABox using RacerPro 2.0, it gives us true for both checks. The result is as expected since the inserted instances comply with the restriction " Disjunctive Mandatory " resulted that each instance of object-type "A" must play at least one of the constrained roles.
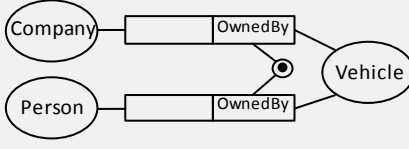

OWL 2 Functional Syntax	Checked Example	
Declaration(Class(:OwnedBy.Company)) EquivalentClasses(:OwnedBy.Company ObjectSomeValuesFrom(:OwnedByC :Company)) Declaration(Class(:OwnedBy.Person)) EquivalentClasses(:OwnedBy.Person ObjectSomeValuesFrom(:OwnedByP :Person)) EquivalentClasses(:Vehicle ObjectUnionOf(:OwnedBy.Company :OwnedBy.Person)) ObjectPropertyDomain(:OwnedByC :Vehicle) ClassAssertion(:Company :jts.com) ClassAssertion(:Person :mira) ClassAssertion(:Vehicle :HYI30) ObjectPropertyAssertion(:OwnedByP :HYI30 :mira) ObjectPropertyAssertion(:OwnedByC :HYI30 :jts.com)		
	Type of Test	**nRQL query**	**Result**
	Coherency Check	? (tbox-coherence? file://disjunctivemandatory.owl)	True
	Consistency Check	? (abox-consistent? file://disjunctivemandatory.owl)	True

Figure 5.6: Tests performed on OWL 2 mapping of ORM disjunctive mandatory constraint (Rule 7)

5.2.8 Role Uniqueness (Rule 8):

An example (Figure 5.7) contains classes Company and Vehicle which are related by the relation OwnedBy is created. The role OwnedBy is constrained Role Uniqueness. An Instance to prove the correctness of formalization is asserted and related by the relation OwnedBy. When we check the coherency of TBox and consistency of ABox using RacerPro 2.0, it gives us true for both checks. The result is as expected since the inserted instances comply with the

restriction "max cardinality of 1" that constraints the role OwnedBy related to the class Vehicle to relate a one instance in the domain to maximum one instance in the range. But when we make an instance like "jts.com" in our example to be related to two instances "MAZ230" and "eng" by the object type property OwnedBy (restricted for the domain "Company" and the range "Vehicle" in our example). The restriction is achieved by using max cardinality of one. Relating one instance of the domain to two instances of the range by object type property "OwnedBy" violates the constraint max cardinality of one. The result is inconsistency of ABox. In the example mentioned above, we use the construct DifferentIndividuals to differentiate between the instances used semantically, where OWL 2 does not support Unique Name Assumption (UNA) [SWM04], which means that using different names for individuals does not mean that these individuals are different.
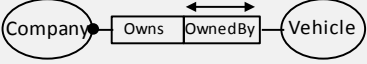
OWL 2 Functional Syntax	Checked Example		
Declaration(Class(:Vehicle)) Declaration(Class(:Vehicle)) EquivalentClasses(:Vehicle ObjectMaxCardinality(1 :OwnedBy :Company)) EquivalentClasses(:Company ObjectSomeValuesFrom(:Owns :Vehicle)) Declaration(ObjectProperty(:OwnedBy)) ObjectPropertyDomain(:OwnedBy :Vehicle) ObjectPropertyRange(:OwnedBy :Company) Declaration(NamedIndividual(:HYI30)) ClassAssertion(:Vehicle :HYI30) Declaration(NamedIndividual(:MAZ230)) ClassAssertion(:Vehicle :MAZ230) Declaration(NamedIndividual(:jts.com)) ClassAssertion(:Company :jts.com) ObjectPropertyAssertion(:OwnedBy :HYI30 :bis.com) DifferentIndividuals(:MAZ230 :HYI30) DifferentIndividuals(:bis.com :HYI30) DifferentIndividuals(:jts.com :MAZ230) ObjectPropertyAssertion(:OwnedBy :HYI30 :jts.com) ObjectPropertyAssertion(:OwnedBy :MAZ230 :jts.com)	**Type of Test**	**nRQL query**	**Result**
	Consistency Check	? (abox-consistent? file:// uniqueness.owl) (with assertions that <u>do not</u> violate the constraint)	True
		? (abox-consistent? file://disjointclasses.owl) (with assertions that <u>do</u> violate the constraint)	Nill

Figure 5.7: Tests performed on OWL 2 mapping of ORM role uniqueness and mandatory constraints (Rule 5)

5.2.9 Role Frequency Constraints (Rule 9):

The same methodology is used as in (5.2.8) to evaluate the correctness of Role Frequency Constraints' mapping. The example in the previous section is modified to represent this rule (Role Frequency Constraints), where this example contains classes "Company" and "Vehicle" related by the relation "Owns". The role "Owns" is played by the object-type "Company" for a

number of occurrences between 2 and 3. This is achieved by restricting "Owns" to play "Vehicle" between 2 and 3 occurrences using the qualified number restrictions of OWL 2 *ObjectMinCardinality* and *ObjectMaxCardinality* constructs. We instantiate "Company" with "ccs.com" and "Vehicle" with four different instances. Relating "ccs.com" with the four instances of "Vehicle" violating the restriction on object property "Owns". When checking the consistency of ABox using Racer reasoned, the reasoned gives inconsistent ABox. This violating of used constraint and the result of inconsistency proves the correctness of using *owl:maxQualifiedCardinality* to map the max occurrence "m" of Role Frequency Constraints (ORM notation).

5.2.10 Value Constraint (Rule 10):

The same general format is used as in Fig. 3.9 to create a complete example which is loaded into RacerPro 2.0. The inserted values for data property range *rB* are X_1, X_2 and X_3. The cases (Fig 5.8) which are used to prove the correctness of Value Constraint formalizing are as follows:

i)   The values of data property range are retrieved and the result is as expected.

ii) One of the assigned values for data type property range is retrieved and the result is true as expected.

iii) A value which is not one of the assigned values for data type range is retrieved and the result is as expected. The result of retrieving this value is an error with a message "Undefined value" that assures that the values of data property range are only those values specified by *DataOneOf* construct.

Type of check	nRQL query	Result
Retrieve values	(retrieve(?x)(?x #rB))	(((:X3))((:X2)) ((:X1)))
Retrieve Check	(retrieve() (:X1 #rB))	True
	(retrieve() (:X4 #rB))	(:ERROR"Undefined value |#X4| in ABox |file://Value.owl|")

Figure 5.8: Cases of queries for checking value constraint correctness (Rule 10)

5.2.11 Subset Role (Rule 11):

An example (Figure 5.9) in OWL 2 that contains classes Person, Vehicle, DrivingLicense, AuthorizedWith.DrivingLicense and Drives.Vehicle, in addition to roles AuthorizedWith and Drives is created. Instances are inserted for classes Person, Vehicle, DrivingLicense. Also instances are inserted for object properties AuthorizedWith and Drives. All these insertions are made to apply instance checking to this ontology to prove the correctness of this ontology that represents the formalizing (formalizing subset role constraint of ORM using OWL 2).
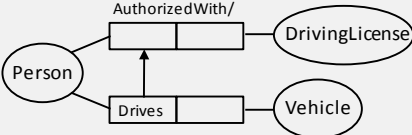
OWL 2 Functional Syntax	Checked Example		
Declaration(Class(:Drives.Vehicle)) EquivalentClasses(:Drives.Vehicle ObjectSomeValuesFrom(:Drives :Vehicle)) SubClassOf(:drives.Person :owns.Person) Declaration(Class(:AuthorizedWith.DrivingLicense)) EquivalentClasses(:AuthorizedWith.DrivingLicense ObjectSomeValuesFrom(:AuthorizedWith :DrivingLicense)) ObjectPropertyDomain(:Drives :Person) ObjectPropertyRange(:Drives :Vehicle) ObjectPropertyDomain(:AuthorizedWith :Person) ObjectPropertyRange(:AuthorizedWith :DrivingLicense) ClassAssertion(:Person :jawad) ClassAssertion(:.Person :sahar) ClassAssertion(:DrivingLicense :LC2011) ClassAssertion(:Vehicle :HYI30) ObjectPropertyAssertion(:AuthorizedWith :jawad :LC2011) ObjectPropertyAssertion(:Drives :sahar :HYI30)			
	Type of Test	**nRQL query**	**Result**
	Retrieve Instances	(retrieve(?x)(?x#Drives. Vehicle))	(:sahar)
		(retrieve(?x)(?x #AuthorizedWith.Drivi ngLicense))	((:jawad) (:sahar))

Figure 5.9: Tests performed on OWL 2 mapping of ORM role subset constraint (Rule 11)

The cases for checking the correctness of formalizing as illustrated in Fig. 5.9 are:

i) The equivalent class Drives.Vehicle is checked to be a subset of equivalent class AuthorizedWith.DrivingLicense and the result is true as expected.

ii) The inserted instance sahar (as an example) to class Person is set to play role 'Drives' and it is checked if it also plays role 'AuthorizedWith' and the result is true as expected.

5.2.12 Subset Binary Role (Rule 12):

An example is created and loaded into RacerPro 2.0 by using its RacerPorter interface. The example includes classes Person and Car, in addition to object properties owns and drives

where the role drives is a subset of the role owns. An instance named ahmad is asserted to class Person and plays the role drives for the instance honda200 of class Car and it is checked if it is at the same time plays the role owns and the result is true as expected (Fig. 5.10).

OWL 2 Functional Syntax	Checked Example		
SubObjectPropertyOf(:AffiliatedWith :WorksFor) SubObjectPropertyOf(:AffiliatedBy :Employes) InverseObjectProperties(:WorksFor :Employes) ObjectPropertyDomain(:WorksFor :Person) ObjectPropertyRange(:ownedBy :Person) InverseObjectProperties(:AffiliatedWith :AffiliatedBy) ObjectPropertyDomain(:AffiliatedWith :Person) ObjectPropertyRange(:WorksFor :Company) ClassAssertion(:Person :ahmad) ObjectPropertyAssertion(:WorksFor :ahmad :jts.com) ClassAssertion(:Person :bassam) ClassAssertion(:Company :jts.com) ClassAssertion(:Company :bsi.com) ObjectPropertyAssertion(:AffiliatedWith :bassam :bis.com)			
	Type of Test	**nRQL query**	**Result**
	Retrieve Instances	(retrieve (?x ?y) (?x ?y #AffiliatedWith))	((:bassam) (:bis.com))
		(retrieve (?x ?y) (?x ?y #WorksFor))	(((:ahmad) (:jts.com)) ((:bassam) (:bis.com)))

Figure 5.10: Tests performed on OWL 2 mapping of ORM binary role constraint (Rule 12)

5.2.13 Equality Role (Rule 13):

An example (Figure 5.11) is created which demonstrates an equality between the equivalent classes expressions HasOfficeWith.Room and HasOfficeIn.Building and one instances are inserted for each equivalent class expression. After testing the instances of the two class expressions, each class expression contains both instances as a result of role equality.

OWL 2 Functional Syntax	Checked Example		
Declaration(Class(:HasOfficeWith.Room)) EquivalentClasses(:HasOfficeWith.Room ObjectSomeValuesFrom(:HasOfficeWith :Room)) Declaration(Class(:HasOfficeIn.Building)) EquivalentClasses(:HasOfficeIn.Building ObjectSomeValuesFrom(:HasOfficeIn :Building)) ObjectPropertyDomain(:HasOfficeWith :Person) ObjectPropertyRange(:HasOfficeWith :Room) ObjectPropertyDomain(:HasOfficeIn :Person) ObjectPropertyRange(:HasOfficeIn :Building) EquivalentClasses(:HasOfficeWith.Room :HasOfficeIn.Building) ClassAssertion(:HasOfficeWith.Room :rajee) ClassAssertion(:Person :rajee) ClassAssertion(:Person :zaher) ObjectPropertyAssertion(:HasOfficeWith :rajee :R101) ObjectPropertyAssertion(:HasOfficeIn :zaher :IT) ClassAssertion(:HasOfficeIn.Building :zaher)			
	Type of Test	**nRQL query**	**Result**
	Retrieve Instances	(retrieve(?x)(?x #HasOfficeWith.Room))	((:rajee) (:zaher))
		(retrieve(?x)(?x #HasOfficeIn.Building))	((:rajee) (:zaher))

Figure 5.11: Tests performed on OWL 2 mapping of ORM equality role constraint (Rule 13)

5.2.14 Equality Binary Role (Rule 14):

An example (Fig. 5.12) is used for the purpose of mapping correctness. This example is written in OWL 2 functional syntax. The example includes classes Vehicle and Person, in addition to object properties Owns (inverse of OwnedBy) and Drives (inverse of DrivenBy) and these roles are set to be equivalent to each other. Instances for classes Person and Vehicle, which are related by drives relation, are inserted.

For this rule (Equality Binary Role), all possible cases (Fig. 5.12) are taken into consideration as follows:

i) The instances of class Person and their consequences of instances of class Vehicle, which are related by the role Owns, are retrieved and, as expected, they are the same as the instances of class Person and their consequences of class Vehicle playing the role Drives.

ii) The inverse case is done where the instances of class Vehicle and their consequences of instances for class Person related by the role drives and owns are retrieved and the result is as expected.

OWL 2 Functional Syntax	Checked Example		
InverseObjectProperties(:drivenBy :drives) ObjectPropertyDomain(:drivenBy :Vehicle) ObjectPropertyRange(:drivenBy :Person) EquivalentObjectProperties(:drives :owns) ObjectPropertyDomain(:drives :Person) ObjectPropertyRange(:drives :Vehicle) InverseObjectProperties(:owns :ownedBy)			
ObjectPropertyDomain(:ownedBy :Vehicle) ObjectPropertyRange(:ownedBy :Person) ObjectPropertyDomain(:owns :Person) ObjectPropertyRange(:owns :Vehicle) ClassAssertion(:Person :ahmad) ClassAssertion(:Vehicle :honda200) ClassAssertion(:Vehicle :volvo100) ClassAssertion(:Person :sahar) ObjectPropertyAssertion(:drives :ahmad :honda200) ObjectPropertyAssertion(:drives :sahar :volvo100)	**Type of Test**	**nRQL query**	**Result**
	Retrieve Instances	(retrieve (?x ?y) (?x ?y #drives))	((((:ahmad) (:honda200)) ((:sahar) (:volvo100)))
		(retrieve (?x ?y) (?x ?y #owns))	((((:ahmad) (:honda200)) ((:sahar) (:volvo100)))
		(retrieve (?x ?y) (?x ?y #drivenBy))	((((:ahmad) (:honda200)) ((:sahar) (:volvo100)))
		(retrieve (?x ?y) (?x ?y #ownedBy))	((((:honda200) (:ahmad)) ((:volvo100) (:sahar)))

Figure 5.12: Tests performed on OWL 2 mapping of ORM equality binary role constraint (Rule 14)

5.2.15 Exclusion Role (Rule 15):

An example in OWL 2 that contains classes Vehicle, Company, Person, OwnedBy.Company and OwnedBy.Person, in addition to roles OwnedByC and OwnedByP. Instances are inserted for classes Vehicle, Company and Person. Also, instances are inserted for object properties OwnedByC and OwnedByP. All these insertions are made to apply instance checking to this ontology to prove the correctness of this ontology that represents the mapping (mapping Exclusive Role from ORM into OWL 2). For Exclusive Role the cases are:

i) The equivalent class OwnedBy.Company is checked to be equivalent to the complement of equivalent class OwnedBy.Person and the result is true as expected.

ii) Consistency check is performed without violating exclusion constraint and the result is as expected consistent Abox with Tbox.

iii) The inserted instance jts.com to class Company is set to play both roles OwnedByC and OwnedByp to vaiolate the exclusion constraint and the result Abox is expected to be inconsistent as a result of role exclusion and the result is as expected (inconsistent Abox).
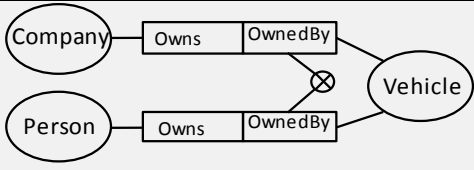
OWL 2 Functional Syntax	Checked Example			
Declaration(Class(:OwnedBy.Company)) EquivalentClasses(:OwnedBy.Company ObjectComplementOf(:OwnedBy.Person)) EquivalentClasses(:OwnedBy.Company ObjectSomeValuesFrom(:OwnedByC :Company)) Declaration(Class(:OwnedBy.Person)) EquivalentClasses(:OwnedBy.Person ObjectSomeValuesFrom(:OwnedByP :Person)) ObjectPropertyDomain(:OwnedByC :Vehicle) ObjectPropertyDomain(:OwnedByP :Vehicle) ClassAssertion(:Person :mira) ClassAssertion(:Company :jts.com) ClassAssertion(:Vehicle :HYI30) ObjectPropertyAssertion(:OwnedByC :HYI30 :jts.com) ObjectPropertyAssertion(:OwnedByP :HYI30 :mira)				
	Type of Test	**nRQL query**		**Result**
	Concept-Equivalence	(concept-equivalent? |#OwnedBy.Company| (not |#OwnedBy.Person|))		True
	Consistency Check	? (abox-consistent? file:// exclusionrole.owl) (with assertions that <u>do not</u> violate the		True
		? (abox-consistent? file://exclusionrole.owl) (with assertions that <u>do</u> violate the constraint)		Nill

Figure 5.13: Tests performed on OWL 2 mapping of ORM exclusion role constraint (Rule 15)

5.2.16 Exclusion Binary Role (Rule 16):

An example is created and loaded into RacerPro 2.0 by using its RacerPorter interface. The example includes classes Person and Car, in addition to object properties owns and drives. An instance named ahmad is asserted to class Person and plays the role drives for the instance honda200 of class Car and at the same time this instance (ahmad) plays the role owns for the same instance honda200 of class Car. This assertion is expected to give inconsistency for Abox and this result is obtained as expected (inconsistent Abox). The result of inconsistency of ABox with TBox proves the mapping of Exclusion Binary Relation using OWL 2 construct (DisjointObjectProperties).

OWL 2 Functional Syntax	Checked Example		
InverseObjectProperties(:Owns :OwnedBy) ObjectPropertyDomain(:Owns :Person) ObjectPropertyRange(:Owns :Vehicle) InverseObjectProperties(:WantsToBuy :BoughtBy) ObjectPropertyDomain(:OwnedBy :Vehicle) ObjectPropertyRange(:OwnedBy :Person) ObjectPropertyDomain(:WantsToBuy :Person) ObjectPropertyRange(:WantsToBuy :Vehicle) ObjectPropertyDomain(:BoughtBy :Vehicle) ObjectPropertyRange(:BoughtBy :Person) DisjointObjectProperties(:Owns :WantsToBuy) ClassAssertion(:Person :ahmad) ClassAssertion(:Vehicle :honda200) ObjectPropertyAssertion(:Owns :ahmad :honda200) ObjectPropertyAssertion(:WantsToBuy :ahmad :honda200)			
	Type of Test	**nRQL query**	**Result**
	Retrieve Instances	(retrieve (?x ?y) (?x ?y #Owns))	((:ahmad)(: honda200))
		(retrieve (?x ?y) (?x ?y #WantsToBuy))	((:ahmad) (:honda200
	Consistency Check	? (abox-consistent? file:// exclusionbinary.owl) (with assertions that <u>do not</u> violate the constraint)	True
		? (abox-consistent? file://exclusionbinary.owl) (with assertions that <u>do</u> violate the constraint)	Nill

Figure 5.14: Tests performed on OWL 2 mapping of ORM exclusion binary role constraint (Rule 16)

5.2.17 Symmetric Ring Constraint (Rule 17):

For Symmetric Ring Constraint, we assign an instance ahmad (Fig. 5.15) for the class Person that plays the role *Likes* for the instance issa, which is also assigned to class Person. We retrieve the instances related by the role likes and the result is as expected ahmad likes issa and issa likes ahmad and this achieves the symmetric constraint.
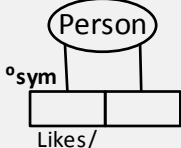
OWL 2 Functional Syntax	Checked Example	Person °sym Likes/		
Declaration(Class(:Person)) Declaration(ObjectProperty(:isLikedBy)) InverseObjectProperties(:isLikedBy :likes) Declaration(ObjectProperty(:likes)) InverseObjectProperties(:isLikedBy :likes) SymmetricObjectProperty(:likes) ObjectPropertyDomain(:likes :Person) ObjectPropertyRange(:likes :Person) Declaration(NamedIndividual(:ahmad)) ClassAssertion(:Person :ahmad) ObjectPropertyAssertion(:likes :ahmad :issa) Declaration(NamedIndividual(:issa)) ClassAssertion(:Person :issa)	**Type of Test**	**nRQL query**		**Result**
	Retrieve Instances	?(retrieve(?x?y)(?x?y #likes))		(((:ahmad) (:issa)) ((:issa) (:ahmad)))

Figure 5.15: Tests performed on OWL 2 mapping of ORM symmetric ring constraint (Rule 17)

5.2.18 Asymmetric Ring Constraint (Rule 18):

For Asymmetric Constraint, an instance ahmad is asserted for the class Person and plays the role *ParentOf* for an instance issa, and at the same time the symmetric relation is created where issa is made to play the role *ParentOf* for the instance ahmad (Fig. 5.16). When checking the consistency for the Abox, it is as expected inconsistent and this is true where the role ParentOf is defined as Asymmetric and as in the example ahmad is parent of issa, then issa cannot be parent of ahmad.
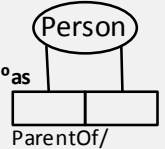
OWL 2 Functional Syntax	Checked Example	Person °as ParentOf/	
Declaration(Class(:Person)) Declaration(ObjectProperty(:SonOf)) InverseObjectProperties(:Son :ParentOf) Declaration(ObjectProperty(:ParentOf)) AsymmetricObjectProperty(:ParentOf) ObjectPropertyDomain(:ParentOf :Person) ObjectPropertyRange(:ParentOf :Person) Declaration(NamedIndividual(:ahmad)) ClassAssertion(:Person :ahmad) ObjectPropertyAssertion(:ParentOf :ahmad :issa) Declaration(NamedIndividual(:issa)) ClassAssertion(:Person :issa) ObjectPropertyAssertion(:ParentOf :issa :ahmad)	**Type of Test**	**nRQL query**	**Result**
	Retrieve Instances	?(retrieve(?x?y)(?x?y #ParentOf))	(((:ahmad) (:issa)) ((:issa) (:ahmad)))
	Consistency Check	? (abox-consistent? file:// asymmetric.owl) (with assertions that <u>do not</u> violate the constraint)	True
		? (abox-consistent? file://asymmetric.owl) (with assertions that <u>do</u> violate the constraint)	Nill

Figure 5.16: Tests performed on OWL 2 mapping of ORM asymmetric ring constraint (Rule 18)

5.2.19 Irreflexive Ring Constraint (Rule 19):

For Irreflexive Constraint, an instance mira of class Person is made to play the role *SisterOf* for instance maya which does not violate the irreflexive constraint and Abox is consistent. Also instance mira is set to play the role *SisterOf* for itself (Fig. 5.17). When checking the consistency for Abox, it gives us inconsistent Abox as expected, which is true since the role *SisterOf* is defined as irreflexive.
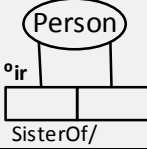
OWL 2 Functional Syntax	Checked Example		
Declaration(Class(:Person)) Declaration(ObjectProperty(:SisterOf)) InverseObjectProperties(:SisterOf : SisterOf) Declaration(ObjectProperty(:SisterOf)) IrreflexiveObjectProperty(:SisterOf) ObjectPropertyDomain(:likes :Person) ObjectPropertyRange(:likes :Person) Declaration(NamedIndividual(:mira)) ClassAssertion(:Person :maya) Declaration(NamedIndividual(:maya)) ClassAssertion(:Person :maya) ObjectPropertyAssertion(:SisterOf :mira :maya) ObjectPropertyAssertion(:SisterOf :mira :mira)	**Type of Test**	**nRQL query**	**Result**
	Retrieve Instances	?(retrieve(?x?y)(?x?y #SisterOf))	(((:mira) (:maya)) ((:mira) (:mira)))
	Consistency Check	? (abox-consistent? file:// irreflexive.owl) (with assertions that do not violate the constraint)	True
		? (abox-consistent? file://irreflexive.owl) (with assertions that do violate the constraint)	Nill

Figure 5.17: Tests performed on OWL 2 mapping of ORM irreflexive ring constraint (Rule 18)

5.2.20 Syntatic Sugar for ORM/OWL 2

5.2.20.1 Identity Constraint

An example in OWL 2 that contains a class named Person with a defined construct HasKey named issn is created (Fig. 5.18). Different instances ahmad and ayser are asserted for the class Person and populated by the data type property issn to the same instance 12345 of type integer. This population of the relation (issn) makes the ABox inconsistent with TBox. The inconsistency of ABox proves that the Haskey construct works as intended from the perspective of machine logic. Here we use Pellet 2.2.2[15] reasoner which supports OWL 2 and HasKey construct.

---

[15] http://clarkparsia.com/pellet/

OWL 2 Functional Syntax	Checked Example	Person (issn)		
Declaration(Class(:Person)) Declaration(DataProperty(:issn)) Declaration(NamedIndividual(:ahmad)) ClassAssertion(:Person :ahmad) DataPropertyAssertion(:issn :ahmad "12345"^^xsd:integer) Declaration(NamedIndividual(:aysar)) ClassAssertion(:Person :aysar) DataPropertyAssertion(:issn :aysar "12345"^^xsd:integer) DifferentIndividuals(:aysar :ahmad) DataPropertyAssertion(:issn :ahmad "12345"^^xsd:integer) DataPropertyAssertion(:issn :aysar "12345"^^xsd:integer) HasKey(:Person () (:issn))	**Type of**	**nRQL query**		**Result**
	Consistensy Check	Pellet Consistency c:/ hasKey.owl) (with assertions that do not violate the constraint)		Yes
		Pellet Consistency c:/ hasKey.owl) (with assertions that do violate the constraint)		No

Figure 5.18: Tests performed on OWL 2 mapping of ORM identity constraint

5.2.20.2 Total and Exclusive Constraints

An example as shown in Figure 5.18, in OWL 2 is created. That contains a class named Vehicle as a super type and other two classes named PrivateCar and VanCar as a subtypes for the class Vehicle. This example is created to prove the correctness of disjoint union formalizing. When retrieving the instances of class Vehicle, it gives us the union of instances of subclasses PrivateCar and VanCar, and this achieves unionOf between subclasses. The instances are made different from each other where OWL 2 does not support unique name assumption. After asserting the same individual mercedes100 to class VanCar, where this instance is already asserted to class Private Car, this assertion violates the disjoitness between classes VanCar and PrivateCar. The result of checking the consistency of ABox is nil (ABox is inconsistent) and this result is as expected.

OWL 2 Functional Syntax	Checked Example	Vehicle		
DisjointUnion(:Vehicle :VanCar :PrivateCar) Declaration(NamedIndividual(:bora100)) ClassAssertion(:Vehicle :bora100) ClassAssertion(:VanCar :honda100) ClassAssertion(:VanCar :mazda200) ClassAssertion(:PrivateCar :mercedes100) ClassAssertion(:PrivateCar :toyota200) ClassAssertion(:VanCar : mercedes100) DifferentIndividuals(:honda100 :bora100) DifferentIndividuals(:mazda200 :bora100) DifferentIndividuals(:mercedes100 :bora100) DifferentIndividuals(:toyota200 :bora100) DifferentIndividuals(:mazda200 :honda100) DifferentIndividuals(:mercedes100 :honda100) DifferentIndividuals(:toyota200 :honda100) DifferentIndividuals(:mercedes100 :mazda200) DifferentIndividuals(:toyota200 :mazda200)		PrivateCar VanCar		
	Type of Test	**nRQL query**	**Result**	
	Retrieve Instances	(retrieve(?x) (?x #Vehicle))	(((:bora100)) ((:honda100)) ((:mazda200)) ((:mercedes100)) ((:toyota200)))	
	Consistency Check	? (abox-consistent? file:// asymmetric.owl) (with assertions that do not violate the constraint)	True	

DifferentIndividuals(:toyota200 :mercedes100)		? (abox-consistent? file://asymmetric.owl) (with assertions that do violate the constraint)	Nill

Figure 5.19: Tests performed on OWL 2 mapping of ORM total and exclusive constraints

Although the tests performed on the OWL 2 mappings of the ORM constructs cannot be theoretically complete, they cover most of the ground (i.e., they are comprehensive). In fact, in our tests we focused on using the boundary analysis techniques of software testing, where we tested boundary or limit conditions of the constraints. An example of such tests is the consistency check performed on the exclusive constraint as discussed in section 5.2.5. Note that the exclusive constraint means that the population of the subtypes constrained by this rule is pairwise distinct (i.e., no assertion can be the instance of two classes). The limit of this constraint can be simply checked by trying to violate the constraint and then checking the consistency of the A-Box. Consider, as another example, the role frequency constraint. Consider an object-type *A* that is restricted to play a role with 3-5 occurrences only (e.g., a teacher that is restricted to teach between 2-4 courses). Testing the limits of such constraint requires testing the minimum occurrence restriction (i.e., 2) in addition to the maximum occurrence restriction (i.e., 4).

5.3 Evaluation Process of extending ORM for Complete representation of OWL 2

Our work of extending the ORM notation to cover all OWL 2 constructs was evaluated by means of a survey. Each construct of the eleven additional constructs added to the ORM notation was represented by three different graphical notations after extensive analysis of the graphical notations currently used in ORM and the rationale behind them. The three new proposed notations were chosen to comply with the existing ORM graphical notations. The available choices for the graphical representation were then put in a form of a survey shown in appendix c. The survey was revised by two computer engineers working at the Palestinian e-government academy and the Ministry of Telecom and IT involved in the development of the Palestinian Interoperability Framework "Zinnar" who developed related ontologies in ORM, one of the master of computing students who works at the SINA institute at Birzeit university who studied object role modeling, description logic and web ontology language through some of the master courses and the supervisor of the this thesis who is an expert in ORM, description logics and OWL 2.

Workshops were held to explain the broader scope of the research, the motivation behind extending ORM, and the rationale behind each possible graphical representation. Thirty one ORM and OWL practitioners participated in the workshops and survey process, twenty one of these practitioners who conducted a workshop where master students of computing at Birzeit university who studied the course of knowledge engineering taught by Dr.Mustafa Jarrar. The workshop participants were at the end of the knowledge engineering course (the course was at fall 2011 semester) after they had studied object role modeling, description logics and web ontology language. The students participated in the workshop had developed several modeling cases and projects using object role modeling and web ontology language. Through the workshop an explanation for students about the objectives and related information of this thesis was conducted, also an explanation about the extending of ORM and the new proposed graphical notations was introduced, in addition to a clarification about each construct of OWL 2 and its three new proposed graphical notations. A discussion concerning the ORM extension was held by the thesis author, thesis supervisor and workshop students. At the end of workshop discussion these practitioners were asked to determine their preferred graphical representation for each construct (by filling the survey). Also the practitioners were asked to add a new proposed graphical notation other than the already three specified ones if they were willing to do so from their own perspective. The results of the survey were then analyzed and the final notations were determined with the highest percent of selection. Ten of the practitioners were conducted individually. Some of these ten survey fillers were practitioners in ORM who developed the stationers ORM modeling cases for the Palestinian e-government project, others were students of master of computing at Birzeit university who studied the course of knowledge engineering that includes related surveyed subjects.

5.3.1 Equivalent Classes:

Equivalent Classes as an OWL 2 construct means that two or more classes constrained with the construct Equivalent Classes are semantically equivalent to each other. In ORM there are no notations that implement the equivalence between two or more objects. The proposed notations to represent Equivalent Classes in ORM are shown in Figure 5.20; also the usage of using each proposed notation is shown in the same figure. Of the 31 knowledgeable persons

surveyed, ten preferred notation (a), five preferred notation (b) and sixteen preferred notation (c). According to the survey result the graphical notation (c) was preferred by 16 of the 31 experts who participated in the evaluation survey (52%). So this graphical notation was used.
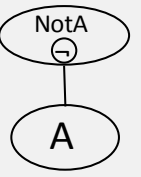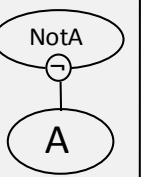

Figure 5.20 Surveyed notations and their percent results for ORM in their symbol and usage shapes to represent OWL 2 construct of Equivalent Classes

5.3.2 Disjoint Classes:

To represent Disjoint Classes construct of OWL 2 in ORM, three notations are suggested as shown in Figure 5.21 according to usually used symbols that logically indicate the semantic of disjoint, also the usage of using each proposed notation is shown in the same figure. Of 31 experts surveyed 20 preferred the ⊗ symbol to represent the Disjoint Classes construct, three preferred the symbol shown in (b) and eight preferred symbol (c). According to the survey result the graphical notation (a) was preferred by 20 of the 31 practitioners who participated in the evaluation survey (64%). So this graphical notation is used.
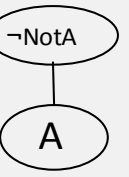

Figure 5.21 Surveyed notations and their percent results for ORM in their symbol and usage shapes to represent OWL 2 construct of Disjoint Classes

5.3.3 Intersection of Class Expressions

Intersection of classes construct of OWL 2 is an expression states that the intersection between classes contains the individuals that are members of each class belongs to intersected classes. ObjectIntersectionOf construct for classes is not represented in ORM. A symbol of intersection (⊓) is proposed to be used for representing the expression of classes' intersection in three different ways as shown in Fig. 5.22. As shown in Fig. 5.22, we represent anonymous class (class with no name) just to show that we have an expression of intersection of classes, which is virtually equivalent to the anonymous class. In implementation we do show the anonymous class graphically. The proposed graphic notation of the circle with the symbol ⊓ attached in different ways within the ellipse represents intersected equivalent class. This proposed graphical notation represents the expression of classes' intersection. The same thing in principle is applied for class complement. Of 31 practitioners surveyed four preferred notation (a), 18 preferred notation (b) and nine preferred notation (c). According to the survey result the graphical notation (b) was preferred by 18 of the 31 experts who participated in the evaluation survey (58%). So this graphical notation is used.


(a)	(b)	(c)
A_Intersection with A$_1$, A$_2$ ... A$_n$	A_Intersection with A$_1$, A$_2$ ... A$_n$	A_Intersection with A$_1$, A$_2$ ... A$_n$
13%	58%	29%

Figure 5.22 Surveyed notations and their percent results for ORM in their symbol and usage shapes to represent OWL 2 construct of Intersected Classes

5.3.4 Class Complement

The Complement of Class expression is not represented in ORM which states that the complement class contains all individuals that are not members of the intended class. The three notations that are proposed to represent the expression of class' complement are shown in Fig. 5.23. The symbol not (¬) is proposed to be used for representing the complement of class

expression as shown in the Fig. 5.23 of 31 practitioners surveyed six preferred notation (a), 14 preferred notations (b) and eleven preferred notation (c). According to the survey result the graphical notation (b) was preferred by 14 of the 31 experts who participated in the evaluation survey (45%). So this graphical notation is used in the implementation.
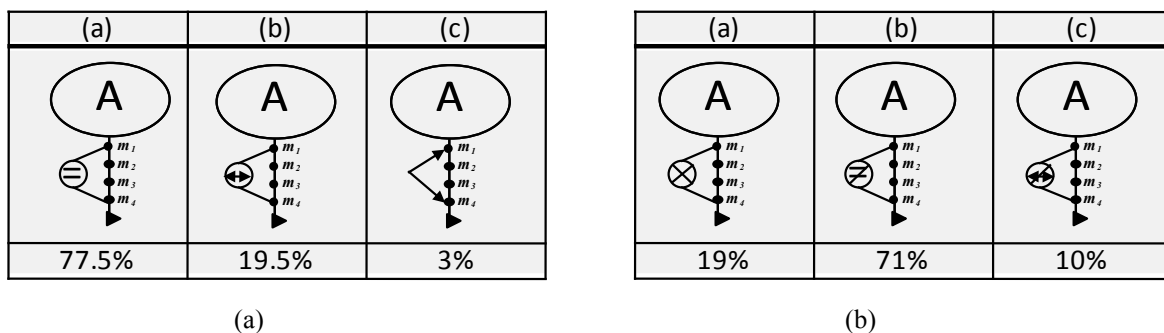

Figure 5.23 Surveyed notations and their percent results for ORM in their symbol and usage shapes to represent OWL 2 construct of Class Complement

5.3.5 Class Assertions

Class Assertions is not formally represented in ORM. The three notations that are proposed to represent class assertions are shown in Fig. 5.24.a. The symbol (►) is proposed to be used to enable the class assertions as shown in Fig. 5.24. Of 31 practitioners surveyed nine preferred notation (a), four preferred notation (b) and fifteen preferred notation (c). According to the result of the survey the graphical notation (c) was preferred by 15 of the 31 practitioners who participated in the evaluation survey (48%). So this graphi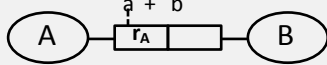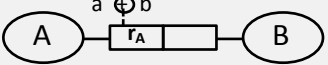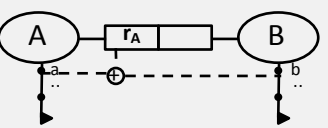cal notation is used in the implementation. Three proposed notations for the appearance of individuals are shown in Fig. 5.23.b. The graphical appearance of individuals is planned to be allowed for four of them, if there is more than four individuals they will be shown within separate window. Of 31 practitioners surveyed 17 preferred notation (a), eight preferred notation (b) and six preferred notation (c). According to the survey result the graphical notation (a) was preferred by 17 of the 31 experts who participated in the evaluation survey (55%).

(a)	(b)	(c)
48%	13%	39%

(a)	(b)	(c)
A	A	A
55%	26%	19%

(a)   (b)
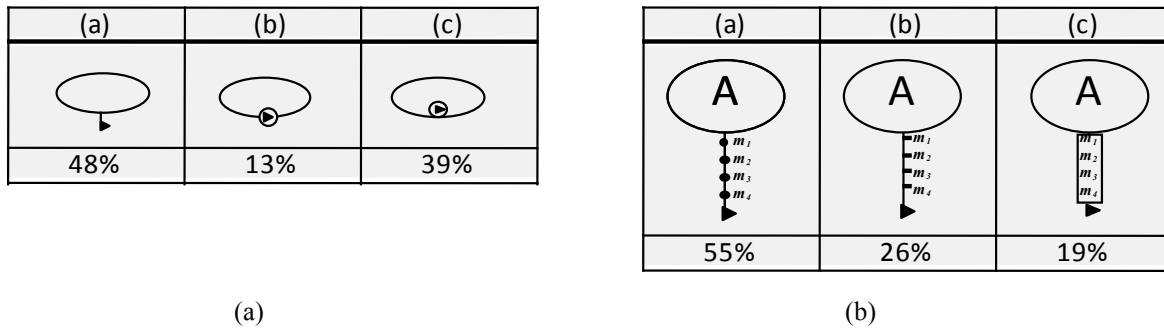
Figure 5.24 Surveyed notations and their percent results for ORM to represent OWL 2 construct of Class Assertions

5.3.6 Individual Equality

Individual Equality states that if we have two instances or more that are related by Individual Equality (OWL 2 Construct), then these instances are the same. The three notations that are proposed to represent Individual Equality are shown in Fig. 5.25.a. Of 31 practitioners surveyed 24 preferred notation in (a), six preferred notation (b) and one preferred notations (c). According to the survey result the graphical notation (a) was preferred by 24 of the 31 experts who participated in the evaluation survey (77.5%). So this graphical notation is used in the implementation.
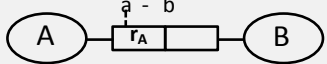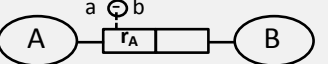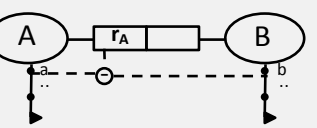
(a)	(b)	(c)
A	A	A
77.5%	19.5%	3%

(a)	(b)	(c)
A	A	A
19%	71%	10%

(a)   (b)

Figure 5.25 Surveyed notations and their percent results for ORM to represent OWL 2 construct of (a) Individual Equality and (b) Individual Inequality

5.3.7 Individual Inequality

Individual Inequality states that if we have two instances or more that are related by Individual Inequality (OWL 2 Construct), then these instances are different from each other. The three notations that are proposed to represent Individual Inequality are shown in Fig. 5.25.b. Of 31 practitioners surveyed six preferred notation in (a), 22 preferred notations (b) and three

preferred notation (c). According to the survey result of the graphical notation (b) was preferred by 22 of the 31 experts who participated in the evaluation survey (71%). So this graphical notation is used in the implementation.

5.3.8 Positive Object/Data Property Assertion

The three notations that are proposed to represent Positive Object Property Assertion are shown in Fig. 5.26. Of 31 practitioners surveyed 19 preferred notation in (a), eight preferred notation (b) and four preferred notation (c). According to the survey result the graphical notation (a) was preferred by 19 of the 31 practitioners who participated in the evaluation survey (61%). So this graphical notation is used in the implementation.

(a)	(b)	(c)
a + b, A — rA — B	a ⊕ b, A — rA — B	A — rA — B, a ⊕ b
61%	26%	13%

Figure 5.26 Surveyed notations and their percent results for ORM to represent OWL 2 construct of Positive Object Property Assertion

5.3.9 Negative Object/Data Property Assertion

The three notations that are proposed to represent Negative Object Property Assertion are shown in Fig. 5.27. Of 31 experts surveyed 21 preferred notation in (a), seven preferred notation (b) and three preferred notation (c). According to the survey result the graphical notation (a) was preferred by 19 of the 31 practitioners who participated in the evaluation survey (68%). So this graphical notation is used in the implementation.

(a)	(b)	(c)
a - b, A — rA — B	a ⊖ b, A — rA — B	A — rA — B, a ⊖ b
68%	23%	9%

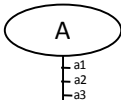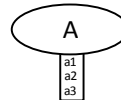Figure 5.27 Surveyed notations and their percent results for ORM to represent OWL 2 construct of Negative Object Property Assertion
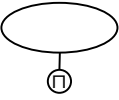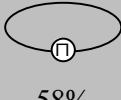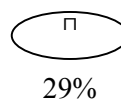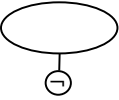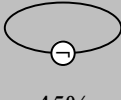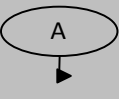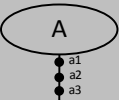
5.3.10 Reflexive Ring Constraint:
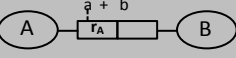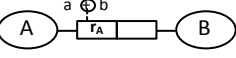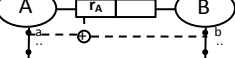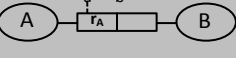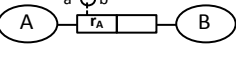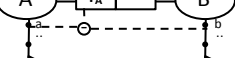
The notation for reflexive constraint is set as that for ring constraints using the word "ref" to indicate the reflexive constraint. This notation was not surveyed because we used exactly the same pattern used in ORM for ring constraints and so as for transitive constraint.

5.3.11 Transitive Ring Constraint:

The notation for reflexive constraint is set as that for ring constraints using the word "tra" to indicate the reflexive constraint.

Table 5.1 summarizes all the graphical representation choices and the results of the survey.

	Construct Name	Representation (1)	Representation (2)	Representation (3)
1	Equivalent Classes	32%	16%	52%
2	Disjoint Classes	64%	10%	16%
3	Intersection of Class Expressions	13%	58%	29%
4	Class Complement	19%	45%	36%
5	Class Assertions (before)	48%	13%	39%
	Class Assertions (after)	55%	26%	19%
6	Individual Equality	77.5%	19.5%	3%

7	Individual Inequality	 71%	 10%	 19%
8	Positive Property Assertion	 61%	 26%	 13%
9	Negative Property Assertion	 68%	 23%	 9%

Chapter Six

Implementation of Mapping between ORM and OWL 2

6.1 Introduction

This chapter provides an overview of the extension of DogmaModeler tool to hold the mapping between ORM and OWL 2. DogmaModeler was developed as a modeling tool using the programming language Java within the integrated development environment Java Beans. DogmaModeler is used as a tool for ontology engineering, where it enables ontology builders to build ontologies that are characterized by both usability and reusability, and by easily using ORM graphical notations [JDM03]. DogmaModeler and its extension can be downloaded from[16]. Using the extended DogmaModeler, the person can graphically build an ontology using graphical representation of objects, relations and constraints on the user-interface for both building and editing the required ontology. The DogmaModeler tool will automatically create the equivalent OWL 2 file that represents the graphically-built ontology using ORM. The mapped OWL 2 file can be checked for correctness (like consistency and coherency checking) using the Hermit reasoner, which is integrated into DogmaModeler.

6.2 DogmaModeler

The work presented in this thesis is implemented as an extension to DogmaModeler. DogmaModeler is an ontology modeling tool based on ORM. DogmaModeler allows modeling, browsing, and managing ontologies. This tool allows one to build his Ontology

---

[16] http://www.jarrar.info/Dogmamodeler/

using the ORM paradigm and then validate his Ontology using integrated Description Logic reasoning services. In its original implementation, DogmaModeler supported the mapping of built ORM ontologies into three notations: ORM-ML (ORM Markup Language), Pseudo Natural Language, and DIG (Description Logic Interface). These mappings allow the ontology to be easily exchanged and understood by domain experts, accessed and processed automatically by application, and validated using Description Logic reasoning services. Originally, ORM diagrams built in DogmaModeler were mapped automatically to DIG; a description logic interface (XML-based language). The DIG mappings of the ORM models were then validated using a Description Logic reasoning server (such as Racer, Fact++, etc) which acted as a background reasoning engine.

In fact, DogmaModeler integrates reasoning services (such as Racer, Fact++, etc) as background reasoning engines. This allows one to validate his/her ontology, for any possible contradiction or inconsistency. The following is a brief description of the three mappings of ORM that DogmaModeler originally support:

(i) *ORM Markup Language (ORM-ML) [J05]*. It is an XML-based language to markup conceptual diagrams, thus allowing the ontology that is built in ORM to be accessed and processed at run-time of applications.

(ii) *Pseudo Natural Language*. Through utilizing ORM's verbalization capabilities, DogmaModeler supports verbalization of ontologies into pseudo natural language, allowing for easy exchange and understanding of the ontologies by domain experts.

(iii) *Description Logic Interface (DIG)*. It is an XML-based language supported by several description logic reasoners (such as Racer, Fact++, etc), which allows for the validation of ontologies built in ORM.

6.3 Extending DogmaModeler

Our implemented extension of DogmaModeler is twofold:

i) We implemented the OWL 2 mapping of ORM presented in this paper, such that the ORM diagrams currently built in DogmaModeler are automatically mapped to OWL 2 and then validated using the Hermit reasoning tool.

ii) We implemented our newly proposed ORM notations with its OWL 2 mapping.

By doing so, one can now build his OWL 2 ontology graphically using ORM and its mapping into OWL 2 is automatically generated and then validated using description logic reasoning. As we mentioned before the idea behind the implementation of our work is enabling ontology building by the intended person using the graphical oriented paradigm.

The DogmaModeler tool is now extended to become an environment for authoring OWL 2 ontologies graphically using ORM. That is, one now builds his OWL 2 ontology graphically using ORM and then DogmaModeler generates the OWL 2 code automatically. This code can then be validated easily using description logic reasoning via the integrated "HermIT" reasoning tool.

For the first part of implementation, we extend DogmaModeler to automatically map ORM into OWL 2 constructs depending on ORM markup language [DJM02, J05] (which is automatically generated according to equivalent ORM graphical notations). Figure 6.1.a shows a snapshot of DogmaModeler outputs, which illustrates an ORM graphical notation example with reasoning results indicating that the built example of ORM is consistent and there is no unsatisfiable concepts. Figure 6.1.b, shows the mapping result of OWL 2 for the example illustrated in Figure 6.1.a.

Fig. 6.2 depicts a modification for ORM diagram of Fig 6.1.a which is built using DogmaModeler. The first window (Fig. 6.2.a) shows the diagram and its validation (by means of logical reasoning) using the integrated HermIT reasoning server while the second window shows the OWL 2 mapping of the diagram. Note that the results of the reasoning state that the class 'ownedBy.Company' is unsatisfiable due to the contradiction between the exclusion and mandatory constraints.
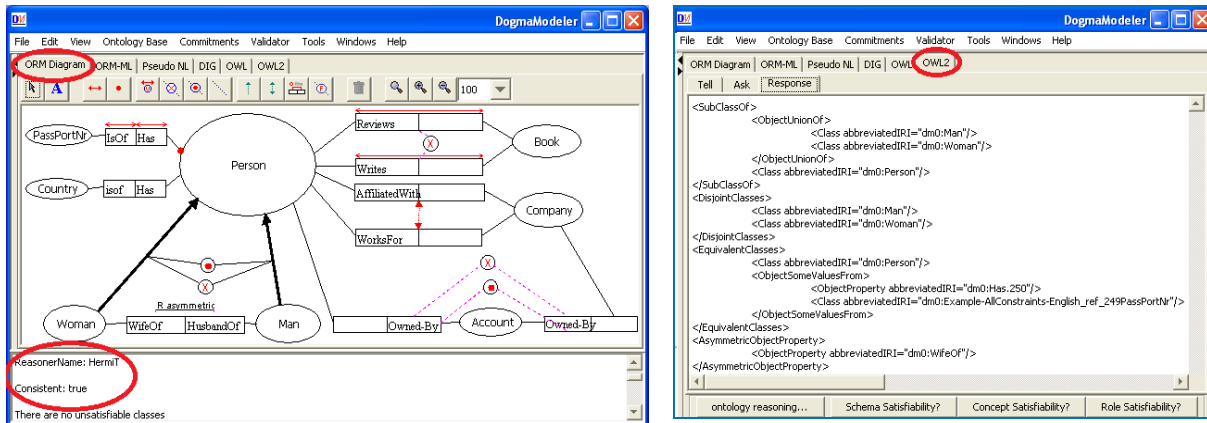
(a) ORM built and validated in DogmaModeler   (b) OWL 2 mapping of the ORM in (a)

Figure 6.1: A Snapshot of an ORM diagram built in DogmaModeler, its mapping to OWL 2, and its validation.
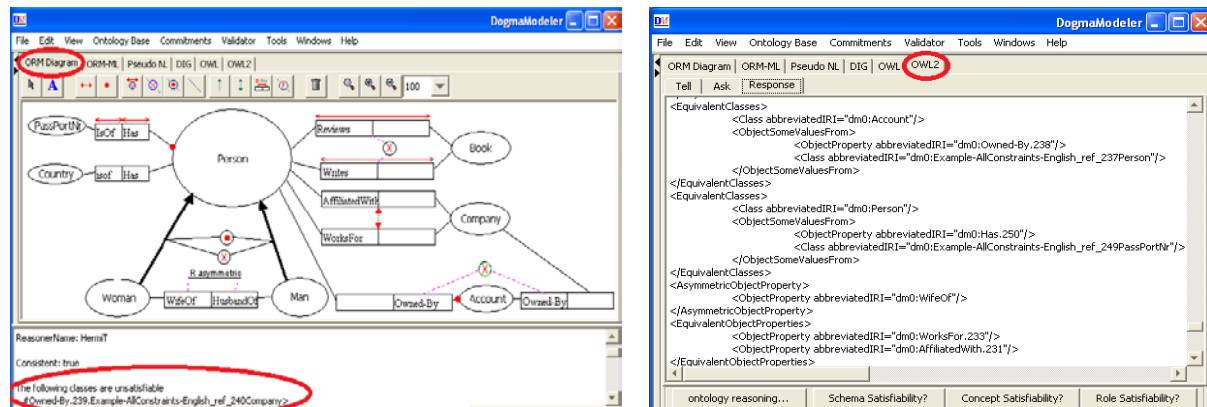
(a) ORM built and validated in DogmaModeler   (b) OWL 2 mapping of the ORM in (a)

Figure 6.2: A Snapshot represent modified ORM diagram of Figure 6.1, built in DogmaModeler, its mapping to OWL 2, and its validation.

For the second part of implementation, that illustrates the extending of ORM notations to represent those of OWL 2 not represented by original ORM notations. We extended DogmaModeler by adding the new proposed ORM notations which are illustrate in chapter 4. These extended notations are integrated with the family of ORM notations holed by DogmaModeler with the full functionality of mapping. The mapping of these new proposed ORM notations is done to both ORM Markup Language and OWL 2. We proposed the new equivalent constructs for ORMML to hold the extended ORM notations, but here our significance concern is mapping the extended ORM notations into OWL 2 that is actually implemented.

As a result of implementation we are mainly concerned with enabling one to graphically build his/her ontology using extended ORM that fully represents of OWL 2. OWL 2 (recommended web ontology language from W3C) contains the majority of constructs to be used for building any needed ontology supported by many reasoners for checking the correctness of built ontology. This extension of ORM constructs is included in the extended DogmaModeler tool, which will enable one to build his ontology graphically without writing OWL 2 syntax and DogmaModeler will automatically convert this ontology to OWL 2 representation file. The resulting file of OWL 2 can be used easily for ontology modeling purposes. One can interactively model the needed ontology using DogmaModeler and after every graphical modeling step she/he will be fed back about his/her progress. The incremental ontology building can be checked continuously for correctness.

Fig. 6.3 shows the use case of Fig. 4.9 (section 4.15) built in DogmaModeler. The first window (Fig. 6.3) contains the ORM diagram. Note that this diagram contains our proposed extension of the ORM notation. The results of the validation using logical reasoning show that the ORM model is consistent and there are not unsatisfiable concepts. The second window (Fig. 6.5.a) shows the OWL mapping of the ORM diagram. Notice the mapping of the newly added ORM notations into OWL 2.
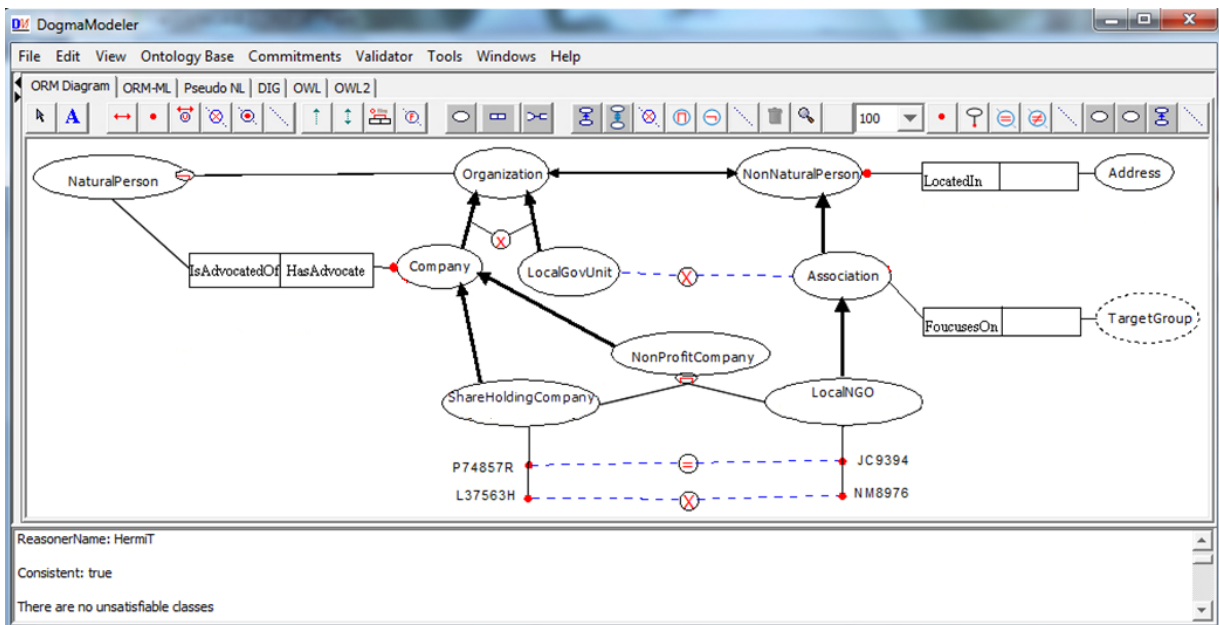

Figure 6.3: ORM use case of Figure 4.9 using DogmaModeler with new proposed ORM notations.

Fig. 6.4 shows another ORM example created using DogmaModeler. The figure depicts three sample ontologies for ministry of labor, health ministry and transportation ministry expressed in ORM; these sample ontologies are integrated with each other using some of the new proposed ORM notations illustrated below. Note that these three sample ontologies are proposed for Palestinian ministries. However, what is depicted in Figure 6.4 is not a complete ontology and is only meant to be a sample demonstration of the usefulness of the newly proposed ORM notations. Fig. 6.5.b shows the equivalent OWL 2 constructs (in OWL/XML syntax) of new proposed ORM notations that are illustrated in Fig. 6.4.
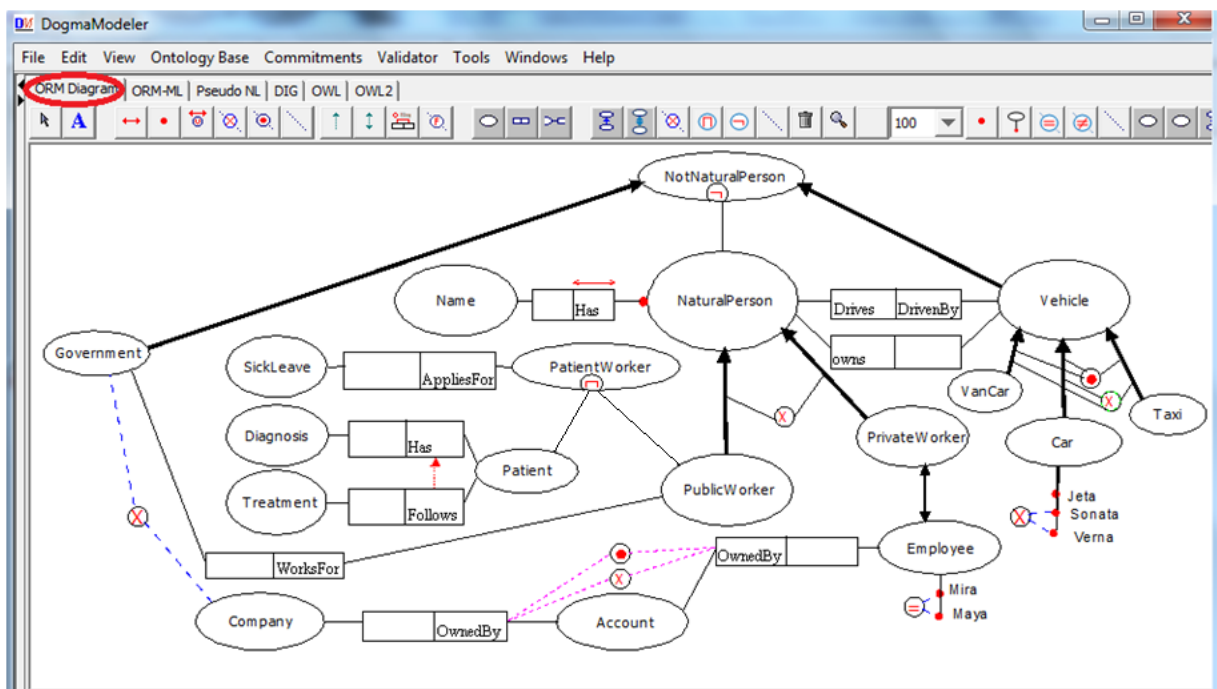
Figure 6.4: ORM example using DogmaModeler with new proposed ORM notations

Figure 6.4 contains new proposed ORM graphical notations. These notations are illustrated in what follows:

- Equivalent Classes is depicted as "↔" between the equivalent classes PrivateWorker and Employee.

- Disjoint Classes is depicted as "⊗" between the disjoint classes Government and Company.

- Intersection of Class Expressions is depicted as "⊓" between the intersected classes Patient and PublicWorker and the proposed equivalent class PatientWorker.

- Class Complement is depicted as "⊖" connected to the intended class NaturalPerson and the proposed equivalent class NotNaturalPerson which is equivalent the complement of class NaturalPerson.

- Individual Equality is depicted as "⊜" between the same individuals Mira and Maya which are instances asserted as instances into the class Employee.

- Individual Inequality is depicted as "⊘" between the different individuals Sonata and Verna.

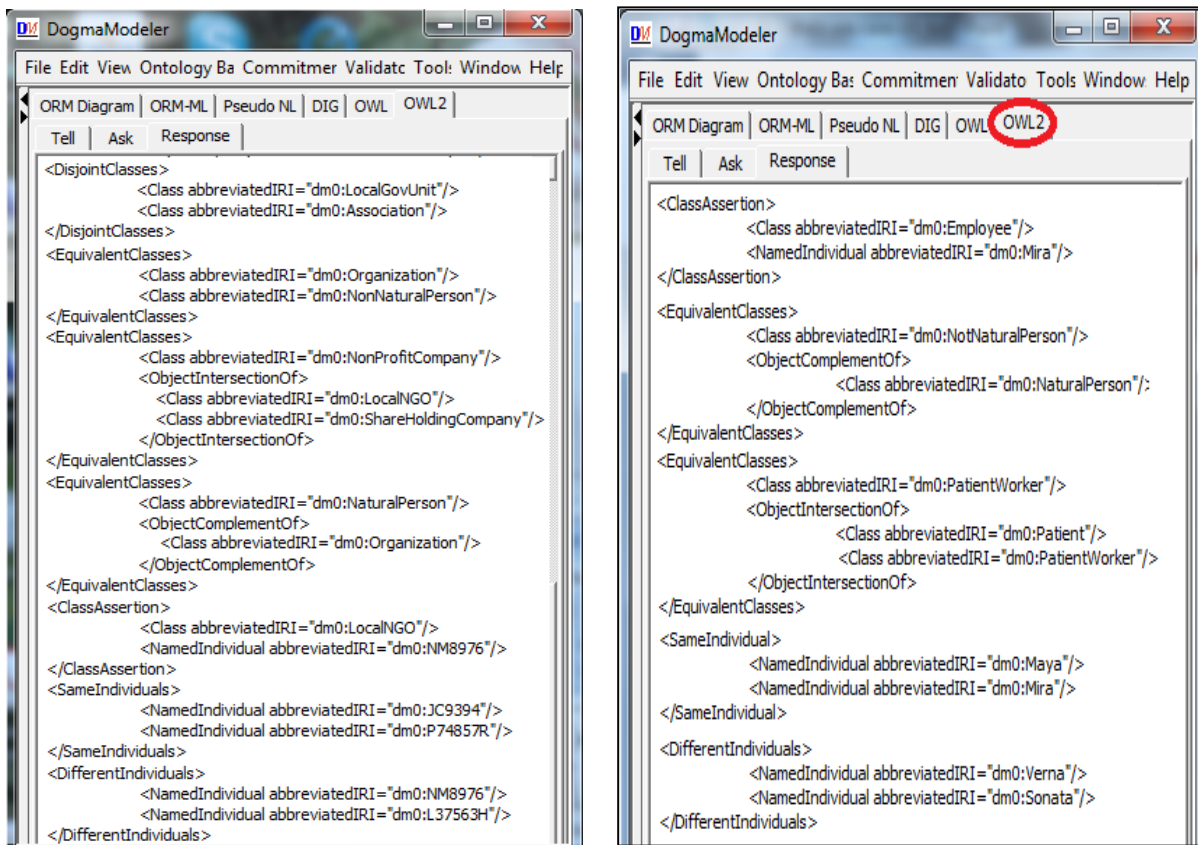Figure 6.5 OWL 2 constructs (in OWL/XML syntax) of new proposed ORM notations that are illustrated in (a) Figure 6.3 and (b) Figure 6.4.

Chapter Seven

Conclusion and Recommendations for Future Work

7.1 Conclusion

The ever increase of the need to develop various applications that depend on ontology pressures the need to build ontology without time and effort consuming. It is more easily for people to perform conceptual modeling graphically. OWL 2 which is the recommended Web Ontology Language needs proficiency and using it to build the needed ontology is time and effort consuming. It is worth to find a way to build ontology graphically and at the same time using OWL 2. ORM which is rich of graphical notations and characterized by high expressivity is used as a conceptual modeling tool in a graphical context.

The mapping and automation of this mapping between ORM and OWL 2 are the main theme of this thesis. We mapped nineteen (out of twenty nine) ORM constructs. The ORM constructs that depend on n-arity cannot be mapped into SROIQ/OWL 2 constructs according to the fact that OWL 2 supports only binary relations. Where these nineteen constructs represent the most commonly used constructs in ORM. At the same time, those constructs are supported by SROIQ Description Logic; which means that OWL 2 output we have mapped is characterized by its ability of decidability. Although the mapping from ORM into OWL 2 is done theoretically using SROIQ Description Logic, which leads to the verification of mapping correctness, we do evaluate this mapping (from ORM into OWL 2). The evaluation here which is done practically using RacerPro reasoner is done to assure practically the correctness

of mapping. This evaluation of mapping from ORM into OWL 2 is done by examining each mapped construct as a complete OWL 2 file (the file is loaded into RacerPro 2.0 reasoner) using different reasoning services such as consistency, coherency and instance checking.

In the second part of this research, we extend ORM for complete representation of OWL 2 . We do proposed new notations of ORM to completely represent OWL 2 by ORM notations, in addition of using non-notational expressions. The proposed notations for extending ORM are selected according to an evaluation process. The evaluation process depends in a survey fulfilled by practitioners in both ORM and OWL.

DogmaModeler as a modeling tool based on ORM is extended to include the new proposed notations of ORM with the already existing notations that are mapped into OWL 2. As a result one can use ORM as an interface of OWL 2 to build him/her ontology graphically, where the built ontology is automatically mapped into OWL 2 using the extended DogmaModeler to perform this mapping (between ORM and OWL 2). Once the mapped OWL 2 file from extended ORM is generated, we can reason about the ontology represented in OWL 2 using Hermit reasoner that supports OWL 2 to check the correctness of the built ontology. It is important to note here that the extended ORM is not merely a graphical notation for the visualization of ontologies. It is a methodology that guides the ontology engineer to design and represent an ontology using the different constructs and rules it provides. ORM facilitates the process of engineering the ontology through its verbalization capabilities which allow the involvement of domain experts in the modeling process.

As a result of this thesis work we have developed an expressive and methodological graphical notation for OWL 2, which allows people to author OWL 2 ontologies graphically. To summarize, we develop our thesis work through two main phases : (i) mapping the graphical notations of ORM to SROIQ/OWL 2 following the semantics of OWL 2 (ii) extending the ORM graphical notation to cover all OWL 2 constructs not currently covered by ORM. OWL 2 is the W3C-recommended ontology language for authoring ontologies. However, it does not provide any methodological or graphical means to engineer ontologies. On the other hand, ORM is a graphical methodological notation used for conceptual modeling. By mapping

between ORM and OWL 2, one can now engineer OWL 2 ontologies graphically using ORM and then map them automatically to OWL 2.

7.2 Recommendations for Future Work

Future directions of our research will involve extending DogmaModeler to allow user-friendly debugging and reasoning that helps the user find the cause of the problem and directions on how to solve it. Note that this is not an implementation issue; it rather needs theoretical research on reasoning problems. In addition, we plan to perform full mapping of ORM into SROIQ/OWL 2, by extending OWL 2 and the underpinning description logic to hold the uncovered notations of ORM. This will include investigating an extension of the SROIQ Description Logic and the development of a new reasoning engine to support complete reasoning of ORM.

References:

[BBC+10] Barzdins J.,Barzdins g.,Cerans K., Liepins R.,and Sprogis A. UML Style Graphical Notation and Editor for OWL 2. SpringerLink, Perspectives in Business Informatics Research. Lecture Notes in Business Information Processing, 2010, Volume 64, Part 2, 102-114, DOI: 10.1007/978-3-642-16101-8_9.

[BCG05] Berardi D., Calvanese D., and Giacomo G. Reasoning on uml class diagrams. Artificial Intelligence, 168(1):70–118, 2005.

[BCM+07] F. Baader D., Calvanese D., McGuinness D., Nardi, and P. F. Patel-Schneider, editors. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2nd edition, August 2007.

[BHH+04] Bechhofer S., Harmelen F. v., Hendler J., Horrocks I., McGuinness D. L., Patel-Schneider P. F., and Stein L. A. OWL Web Ontology Language reference. W3C Recommendation, 10 February 2004. Available at http://www.w3.org/TR/owl-ref/.

[BGME07] Bouras A., Gouvas P., Mentzas G.. ENIO. September 2007. An Enterprise Application Integration Ontology. 18th International Workshop on Database and Expert Systems Applications.

[BKMP09] Bao J., Kendall E., McGuinness D., and Patel-Schneider P. eds. OWL 2 Web Ontology Language: Quick Reference Guide. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-quick-reference-20091027/. Latest version available at http://www.w3.org/TR/owl2-quick-reference/.

[BMST07] Bach D., Meersman R., Spyns P., and Trog D. Mapping owl-dl into orm/ridl. In OTM 2007 Workshops, proceeding of ORM'07. Springer Verlag, November 2007.

[BVEL04] Brockmans S. , Volz R., Eberhart A., Loffler P. Visual modeling of OWL DL ontologies using uml. ISWC 2004, LNCS 3298, pp. 198–213, 2004. Springer-Verlag Berlin Heidelberg 2004.

[CH05] Cuyler D., and Halpin T. 2005. Two Meta-Models for Object-Role Modeling, Information Modeling Methods and Methodologies, eds . Idea Publishing Group, Hershey PA, USA , 17-42.

[CHD+02] Cranefield P. S., Hart L., Dutra M., Baclawski K., Kokar M., and Smith J. 2002. Uml for ontology development. Knowl. Eng. Rev., 17(1):61–64.

[COL08] Conroy C., O'Sullivan, D., and Lewis D. 2008. Ontology Mapping Through Tagging. International Conference on Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. Barcelona, p 886 − 891, ISBN: 978-0-7695-3109-0. IEEE Explore.

[CW05] Cui W., and Wu H. Using ontology to achieve the semantic integration and interoperation of GIS. Geoscience and Remote Sensing Symposium, 2005. IGARSS '05. Proceedings. 2005 IEEE International, ISBN: 0-7803-9050-4.

[DFV03] Davies J., Fensel D., and Van H. F. (eds.) (2003): Towards the semantic web: ontology-driven knowledge management. Wiley, New York.

[DJF11] Deik A., Jarrar M., and Faraj B. Ontology-based Data and Process Governance Framework -The Case of e-Government Interoperability in Palestine, in: In Pre-proceedings of the IFIP International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA'11), Campione, Italy, 2011, pp. 83-98.

[DJM02] Demey J., Jarrar M., and Meersman R. June, 2002. A Markup Language for ORM Business Rules. (RuleML 2002). Volume 60 of CEUR Workshop Proceedings, 107-128, CEUR-WS.org.

[F03] Franconi E. Description Logics for Conceptual Design ,Information Access, and Ontology Integration: Research Trends. 2003, SWIF–ISSN 1126–4780, http://www.swif.uniba.it/lei/ai/networks/.

[F04] Fensel D. (2004): Ontologies: a silver bullet for knowledge management and electronic commerce. Springer, Berlin.

[FFT10] Fillottrani P., Franconi E., and Tessaris S. The ICOM 3.0 Intelligent Conceptual Modelling tool and methodology. Semantic Web – Interoperability, Usability, Applicability 0 (2010) 1–10 1, IOS Press.

[FH11] Fonou-Dombeu J., and Huisman M. 2011. Combining Ontology Development Methodologies and Semantic Web Platforms for E-government Domain Ontology Development. International Journal of Web & Semantic Technology (IJWesT) Vol.2, No.2, pages 12-25, April 2011.

[FN00] Franconi E., and Ng G. The ICOM tool for intelligent conceptual modelling. In Proc. of the 7 th International Workshop on Knowledge Representation meets Databases (KRDB'2000), pages 45–53, 2000.

[G98] Guarino N. 1998. Formal ontologies and information systems. Proceedings of FOIS. Amsterdam, IOS Press (June, 1998), 3-15.

[GH08] Guizzardi G., and Halpin T. A. 2008. Ontological foundations for conceptual modelling. Applied Ontology 3 (1-2): 1-12.

[GW09] Golbreich, C.; and Wallace, E. OWL 2 Web Ontology Language: New Features and Rationale. W3C Editor's Draft, 20 April 2009, http://www.w3.org/2007/OWL/draft/ED-owl2-new-features-20090420/. Latest version available at http://www.w3.org/2007/OWL/draft/owl2-new-features/.W3C Editor's Draft, 20 April 2009, http://www.w3.org/2007/OWL/draft/owl2-new-features/.

[H89] Halpin T. A logical analysis of information systems: static aspects of the data oriented perspective. PhD thesis, University of Queensland, Brisbane, Australia, 1989.

[H01] Halpin T. 2001. Information Modeling and Relational Databases From Conceptual Analysis to Logical Design. Morgan Kufmann.

[H04a] Halpin T. 2004. Business Rule Verbalization, Information Systems Technology and its Applications. Proceedings of ISTA-2004, vol. P-48, 39-52.

[H04b] Halpin T. 2004. Object-Role Modeling: an overview. Microsoft Corporation.

[HC06] Halpin, T., and Curland, M. (2006). Automated Verbalization for ORM 2. OTM'06 Workshops. Springer LNCS. 2006.

[HJ10] Hodrob R., and Jarrar M. Mapping ORM into OWL 2. In proceedings of the International Conference on Intelligent Semantic Web – Applications and Services. Pages 68-73. ACM ISBN 9781450304757. June 2010.

[HKP+09] Hitzler P., Krötzsch M., Parsia B., Patel-Schneider P., and Rudolph S. OWL 2 Web Ontology Language Primer. W3C Editor's Draft 21 September 2009, http://www.w3.org/2007/OWL/draft/ED-owl2-primer-20090921/ Latest version available at http://www.w3.org/2007/OWL/draft/owl2-primer/.

[HO01] Haarslev V. and Oller R. M. RACER System Description. In R. Gor_e, A. Leitsch, and T. Nipkow, editors, Proc. of the 1st Int. Joint Conf. on Automated Reasoning (IJCAR 2001), volume 2083 of LNAI, pages 701-706, Siena, Italy, June 18-23 2001. Springer.

[HS10] Happel H., and Seedorf S. 2010. Applications of Ontologies in Software Engineering. Journal on Collaborative Software Engineering, ISBN: 978-3-642-10293-6. Spriger(2010). http://www.springer.com/978-3-642-10293-6.

[HST99] Horrocks I., Sattler U., and Tobies S. 1999. Practical Reasoning for Expressive Description Logics. In Ganzinger,H.; McAllester, D.; and Voronkov, A., eds., Proc. Of LPAR-6, volume 1705 of LNAI, 161–180. Springer.

[HST06] Horrocks I., Sattler U., and Tobies S. The Even More Irresistible SROIQ. In Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006), pages 57-67. AAAI Press, 2006.

[J05] Jarrar M. May, 2005. Towards Methodological Principles for Ontology Engineering. PhD thesis,Vrije Universiteit Brussel, Brussels, Belgium.

[J07a] Jarrar, M. Towards Automated Reasoning on ORM Schemes. -Mapping ORM into the DLR_idf description logic. In proceedings of the 26th International Conference on Conceptual Modeling (ER 2007). Pages (181-197). LNCS 4801, Springer. Auckland, New Zealand. ISBN 9783540755623. November2007.

[J07b] Jarrar M. Mapping ORM into the SHOIN/OWL Description Logic – Towards a Methodological and Expressive Graphical Notation for Ontology Engineering. OTM Workshops (ORM'07),LNCS480 Springer. http://www.jarrar.info/Publications/. (2007).

[JDM03] Jarrar M., Demey J., and Meersman R. 2003. On using conceptual data modeling for ontology engineering. Journal on Data Semantics (2003).

[JKD06] Jarrar M., Keet M., and Dongilli P. (2006). Multilingual verbalization of ORM conceptual models and axiomatized ontologies. Technical Report. Vrije Universiteit Brussel, Belgium. 2006.

[JM08] Jarrar M. and Meersman R. Ontology Engineering -The DOGMA Approach. Book Chapter in "Advances in Web Semantics I". Chapter 3. Pages 7-34. LNCS 4891, Springer.ISBN:978-3540897835. (2008)

[JMY99] Jurisica I., Mylopoulos J., and Yu E. Using ontologies for knowledge management: an information systems perspective. In: Proceedings of the 62nd Annual Meeting of the American Society for Information Science (ASIS '99), 482–496.

[K07] Keet, C.M. Prospects for and issues with mapping the Object-Role Modeling language into DLR$_{ifd}$. 20th International Workshop on Description Logics (DL'07), 8-10 June 2007, Bressanone, Italy. CEUR-WS Vol-250 / (ISBN 978-88-6046-008-5), 331-338.

[KBB+09] Kendall E., Bell R. , Burkhart R., Dutra M., and Wallace E. Towards a Graphical Notation for OWL 2. Proceedings of OWL: Experiences and Directions 2009 (OWLED 2009), Rinke Hoekstra and Peter F. Patel-Schneider, editors. http://www.webont.org/owled/2009.

[KWF07] Krivov S., Williams R. , and Ferdinando. GrOWL: A tool for visualization and editing of OWL ontologies. Web Semantics: Science, Services and Agents on the World Wide Web 5 (2007) 54–57.

[LH07]Lin H., Harding J . A manufacturing system engineering ontology model on the semantic web for inter-enterprise collaboration. Computers in Industry, Volume 58 Issue 5, June, 2007.

[LSS11] Lezcano L., Sicilia M., and Solano C. Integrating reasoning and clinical archetypes using OWL ontologies and SWRL rules. Journal of Biomedical Informatics 44 (2011), pages 343–353.

[MGH+09] Motik B., Grau B., Horrocks I., Wu Z., Fokoue A., and Lutz C. eds. OWL 2 Web Ontology Language: Profiles. W3C Editor's Draft, 20 April 2009, http://www.w3.org/2007/OWL/draft/ED-owl2-profiles-20090420/. Latest version available at http://www.w3.org/2007/OWL/draft/owl2-profiles/.

[MLL06] Ma Y., Lin Z. and Lin Z., Inferring with Inconsistent OWL DL Ontology: A Multi-valued Logic Approach. Lecture Notes in Computer Science, 2006, Volume 4254/2006, 535-553, DOI: 10.1007/11896548_40.

[MMH04] McGuinness D., McGuinness, and Harmelen F. V. 2004. OWL Web Ontology Language Overview. W3C Recommendation.(Feb.,2004).http://www.w3.org/TR/2004/REC-owl-features-20040210.

[MPP09] Motik B., Patel-Schneider P., and Parsia B. eds. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax W3C Editor's Draft, 20 April 2009, http://www.w3.org/2007/OWL/draft/EDowl2-syntax-20090420/. Latest version available at http://www.w3.org/2007/OWL/draft/owl2-syntax/.

[NB02] Nardi D., and Brachman R. J. An Introduction to Description Logics. In the Description Logic Handbook, edited by F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, Cambridge University Press, 2002. http://www.inf.unibz.it/~franconi/dl/course/dlhb/dlhb-01.pdf.

[PP04] Parsia B., and Pellet E. S. An OWL-DL Reasoner. Poster, In Proc. of the 3rd Int. Semantic Web Conference (ISWC 2004), Hiroshima, Japan, November 7-11, 2004.

[SWM04] Smith M. K., Welty C., and McGuinness D. L. February, 2004. OWL Web Ontology Language Guide.

[TH06] Tsarkov D., and Horrocks I. FaCT++ Description Logic Reasoner: System Description. In Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR 2006), volume 4130 of LNAI, pages 292-297, Seattle, WA, USA, August 17-20 2006. Springer.

[VHS10] Veal D., Hartman N., and Springer J. 2010. Implementing Ontology-based Information Sharing in Product Lifecycle Management. Proceedings of 65th Midyear Conference Engineering Design Graphics Division of ASEE October 3 − 6, 2010 Houghton, Michigan.

[ZLW08] Zhai J., Li J., and Wang O., Using ontology and XML for semantic integration of electricity information systems. Electric Utility Deregulation and Restructuring and Power Technologies, 2008. DRPT 2008. Third International Conference, ISBN: 978-7-900714-13-8.