UNIVERSITE DE NICE-SOPHIA ANTIPOLIS - UFR Sciences
Ecole Doctorale STIC
Sciences et Technologies de l'Information et de la Communication

THESE

pour obtenir le titre de

DOCTEUR EN SCIENCES
de l'UNIVERSITE de Nice-Sophia Antipolis

Spécialité: Informatique

présentée et soutenue par

**Thi Dieu Thu NGUYEN**

# A DL-BASED APPROACH TO INTEGRATE RELATIONAL DATA SOURCES INTO THE SEMANTIC WEB

Thèse dirigée par **Nhan LE-THANH**
soutenue le 5 mai 2008

**Jury:**

| | | |
|---|---|---|
| M. Michel RIVEILL | Professeur à l'UNSA | Président |
| M. Jacques LE MAITRE | Professeur à l'Université du Sud Toulon-Var | Rapporteur |
| M. Franck MORVAN | Maître de Conf. à l'Université de Toulouse 3 | Rapporteur |
| Mme. Zohra BELLAHSENE | Professeur à l'Université de Montpellier II | Examinateur |
| M. Peter SANDER | Professeur à l'UNSA | Examinateur |
| M. Nhan LE-THANH | Professeur à l'UNSA | Directeur |

# Abstract

The Semantic Web is a new Web paradigm that provides a common framework for data to be shared and reused across applications, enterprises and community boundaries. The biggest problem we face right now is a way to "link" information coming from different sources that are often heterogeneous syntactically as well as semantically. Today much information is stored in relational databases. Thus data integration from relational sources into the Semantic Web is in high demand.

The objective of this thesis is to provide methods and techniques to address this problem. It proposes an approach based on a combination of ontology-based schema representation and description logics. Database schemas in the approach are designed using ORM methodology. The stability and flexibility of ORM facilitate the maintenance and evolution of integration systems. A new web ontology language and its logic foundation are proposed in order to capture the semantics of relational data sources while still assuring a decidable and automated reasoning over information from the sources. An automatic translation of ORM models into ontologies is introduced to allow capturing the data semantics without laboriousness and fallibility. This mechanism foresees the coexistence of others sources, such as hypertext, integrated into the Semantic Web environment.

This thesis constitutes the advances in many fields, namely data integration, ontology engineering, description logics, and conceptual modeling. It is hoped to provide a foundation for further investigations of data integration from relational sources into the Semantic Web.

**Keywords**: Data Integration, Semantic Web, Web Ontology Languages, Description Logics, Automated Reasoning, Tableau Algorithms, Conceptual Modeling.

# Résumé

Le web sémantique est un nouveau paradigme web qui fournit un cadre commun pour des données devant être partagées et réutilisées à travers des applications, en franchissant les frontières entre les entreprises et les communautés. Le problème majeur que l'on rencontre à présent, est la manière de relier les informations provenant de différentes sources, celles-ci utilisant souvent une syntaxe et une sémantique différentes. Puisqu'aujourd'hui, la plupart des informations sont gardées dans des bases de données relationnelles, l'intégration de source de données relationnelle dans le web sémantique est donc très attendue.

L'objectif de cette thèse est de fournir des méthodes et des techniques pour résoudre ce problème d'intégration des bases de données. Nous proposons une approche combinant des représentations de schémas à base d'ontologie et des logiques de descriptions. Les schémas de base de données sont conçus en utilisant la méthodologie ORM. La stabilité et la flexibilité de ORM facilite la maintenance et l'évolution des systèmes d'intégration. Un nouveau langage d'ontologie web et ses fondements logiques sont proposées afin de capturer la sémantique des sources de données relationnelles, tout en assurant le raisonnement décidable et automatique sur les informations provenant des sources. Une traduction automatisée des modèles ORM en ontologies est introduite pour permettre d'extraire la sémantique des données rapidement et sans faillibilité. Ce mécanisme prévoit la coexistence d'autre sources d'informations, tel que l'hypertexte, intégrées à l'environnement web sémantique.

Cette thèse constitue une avancée dans un certain nombre de domaine, notamment dans l'intégration de données, l'ingénierie des ontologies, les logiques de descriptions, et la modélisation conceptuelle. Ce travail pourra fournir les fondations pour d'autres investigations pour intégrer les données provenant de sources relationnelles vers le web sémantique.

**Mots-clés**: Intégration de données, Web sémantique, Langages d'ontologie web, Logiques de Descriptions, Raisonnement automatique, Algorithmes de Tableau, Modélisation conceptuelle.

# Contents

# List of Tables

# List of Figures

CHAPTER 1

# Introduction

## 1.1 Motivation

The information society requires full access to the available information, which is often distributed and heterogeneous. Traditional information systems, as well known, are built for that purpose by using some data models and databases. That means to access information from other sources (or systems), an information system must transfer the data formats of these sources to hers. This process is time-consuming and not always easy. Besides, data acquisition (although automatic or semi-automatic) requires inevitably a large investment on the technical infrastructure and/or manpower. The purpose of the *integration* is therefore to create a common gateway to sources heterogeneous and probably distributed.

Continually, new applications are introduced in enterprises, while existing legacy applications cannot be replaced because of the investments that have been made in the past. These new applications need to use data residing in the legacy applications, which is typically stored in a proprietary format. If a new application is to reuse the information residing in legacy systems, this information needs to be made available in a way that is understandable to the new application. Therefore, besides the differences in hardware and software platforms, not only the differences in syntax, but also the differences in semantics need to be overcome. Since systems become more and more distributed and disparate within and across organizational boundaries and market segments, *semantic integration* has become a much-debated topic and it is viewed as a solution provider in both industrial and academic settings.

### 1.1.1 Challenges of Semantic Integration into the Semantic Web

The emergence of the Semantic Web, and its anticipated industrial uptake in the years to come, has made the sharing of information easier to implement and deploy on a large scale in open distributed environments. Since much information on the web is generated from relational databases (the "deep web"), the recent call for the Semantic Web provides additional motivation for the problem of associating semantics with database-resident data. Therefore, it will be necessary to make existing database content available for emerging Semantic Web applications, such as web agents and services.

However, semantic integration of relational data sources into the Semantic Web is not a trivial task. There are (at least) three crucial issues must be considered:

- First, in the foreseeable future, databases, knowledge bases, the World Wide Web, and the Semantic Web will coexist. A challenge for current systems is to accommodate their contents, which are either defined by schemas based on structure or defined by ontologies based on semantics. Therefore, the integration should permit to handle heterogeneous sources generated by not only different data models for databases but also formalisms for other sources, e.g. hypertext. Besides, integration systems should have the capability of evolution and maintenance independently from the run-time characteristics of the underlying sources.

- Second, the design, exploitation and maintenance of such an integration system require reasoning support over information from all those sources. However, database research mainly deals with efficient storage and retrieval with powerful query languages, and with sharing and displaying large amounts of documents while reasoning over the structure and the content of documents plays only a minor role.

- Third, the automation of processing data sources requires to clarify and formalize the knowledge residing in databases. That is, there should have a mechanism specifying the communication/exchange protocol, the vocabularies used and the interpretation rules so that the data semantics can be captured correctly.

### 1.1.2 Objectives and Approach

The aim of this thesis is, therefore, to provide methods and techniques to solve the three above-mentioned issues. This aim is further developed into the following objectives:

1. To integrate resources from databases into the Semantic Web while guaranteeing the semantic integration ability from various resources of the Semantic Web and the evolution and maintenance independently from the run-time characteristics of data sources. We focus on how to make database content available on the Semantic Web.

2. To identify the data model for databases that facilitates the integration regarding the presented issues.

3. To add additional deductive capabilities to integration systems to increase the usability and reusability of data from relational databases.

4. To provide a mechanism to achieve the semantic transparence between data sources and the integration system. The knowledge representation of such a system should be unambiguous, supported by an inference engine, compatible to the standards of World Wide Web and at the same time should permit to capture the semantics of relational data sources.

To achieve those objectives, our approach is to combine ontology-based schema representation and description logics. In particular, our solutions are as follows.

- We use Semantic Web ontologies to incorporate database schemas. That is, data models of databases will be represented in Semantic Web ontologies. In philosophy, an ontology is the study of existence. It seeks to describe the basic categories and relationships of existence to define entities and types of entities within its framework. The Artificial Intelligence and Web experts use this term to describe a document that defines formally relationships between the terms. Ontology plays an important role in the Semantic Web. It represents knowledge in the Semantic Web and is supposed to hold application-independent domain knowledge while conceptual schemas are developed only for the use of enterprise applications. The benefit of representing data models in ontologies is twofold. First, it allows restructuring and facilitating access to the database information, as in the conventional context of Database Management Systems. Second, it permits the use of a

Web-based ontology language as a common language and hence, a range of advanced semantic technologies based on ontologies can be applied to data integration. This solution will help achieving the first objective;

- We propose using ORM data models. ORM is a methodology for conceptual data modeling based on facts. It has a high expressivity and the capability of evolving without breaking the backward compatibility. It has been fully formalized in First Order Logic, which is very closed to the logic in our approach. This choice will help achieving objective 1 and 2;

- We suggest an automated reasoning for description logics. Description logics [7] are a family of knowledge representation formalisms. It can provide a high expressivity while guaranteeing the decidable reasoning, and that, therefore, has been chosen as the logic foundation for many ontology languages as well as for the current standard Web ontology language OWL. OWL is one of the most expressive Web ontology language that is world-wide used today. As a result, we propose using OWL to define the structure, semantics of data resources. By this way, reasoning can be effected over information from relational data sources. This approach is to achieve objectives 3 and 4.

## 1.2 Scope and Limitations

This thesis brings together two large disciplines, namely the field of Description Logics applied to Ontology Engineering in the Semantic Web and the field of Conceptual Data Modeling for databases. This combination is introduced into the field of data integration. However, this thesis does not tackle the following problems, among others:

- *Ontology mapping.* Roughly, ontology mapping is to establish, either manually or semi-automatically, correspondences (i.e. mappings) between different ontologies. Typically, those mappings are useful for data translation and query answering on the ontology layer. However, we focus on making the semantics of data sources available to be exploited by integration systems.

- *Integration from databases not modeled by ORM.* The work presented here is focused on capturing the data semantics residing in ORM schemas. We do not deal with other data models such as relational models, ER or UML diagrams. Interestingly, as we will see in Chapter 1, those schemas can be converted into ORM schemas and vice versa. As a result, our work can be applied to integrate databases from such kinds of model.

- *Ontologies in integration systems not in decidable OWL species.* We represent the semantics of data sources in OWL ontologies (a decidable OWL species). Hence, the integration systems who would like to integrate databases should handle their respective OWL ontologies.

## 1.3 Thesis Outline

The remainder of the thesis is organized as follows.

Chapter 2 introduces the background of data integration. It explains the importance of conceptual modeling and reasoning in such process. Then it analyses conceptual modeling for relational databases so that a relevant data model is identified for our solution.

Chapter 3 introduces the Semantic Web, the environment into which relational data sources are integrated. It shows the importance of description logic based ontologies in the Semantic Web, presents OWL and the description logic theory. Most importantly, the chapter presents a survey of data integration from relational data sources into the Semantic Web. This is the original work introduced in this thesis. The survey clarifies the issues that our approach is going to address in the rest of the thesis.

Chapter 4 proposes a new OWL language, namely OWL-K, in order to capture one of the most important constraints in conceptual modeling, namely identification constraint. This work is motivated by the analysis showing that OWL DL, a decidable OWL species, cannot represent this constraint. Extending OWL DL to OWL-K is, therefore, necessary to capture the semantics of relational data sources.

Chapter 5 proposes an automated reasoning for OWL-K, showing that this Web ontology language is decidable. This feature is very important because it guarantees the deductive capabilities of data integration systems over information from relational databases.

Chapter 6 presents our mechanism to achieve the semantic transparence between data sources and integration systems through the formalization of ORM into OWL-K. This mechanism permits to capture the original semantics of databases and is shown to be automatized by an algorithm and the mapping rules which semantically translate ORM schemas into OWL(-K) ontologies.

Chapter 7 describes how to implement our approach in data integration systems, illustrates our tool, which is at the first stage, to integrate relational data sources into the Semantic Web.

Chapter 8 concludes the thesis. We review the work presented in the thesis and summarize our main contributions. We also point out the open problems and so the perspectives for future research.

The appendixes provide additional information of the implementation in Chapter 7.

CHAPTER 2

# Data Integration and Conceptual Data Modeling

Data integration is a pervasive challenge faced in applications that need to query data across multiple autonomous and heterogeneous data sources. Data integration is crucial in large enterprises that own a multitude of data sources, in large-scale scientific projects where data sets are being produced independently by multiple researchers, for better cooperation among organizations, each with their own data sources, and in offering good search quality across millions of structured data sources, such as relational databases, on the World Wide Web.

Data sources are heterogeneous on the syntactical and semantic levels. The *syntactical level* encompasses the way the *data model* is written down. A data model is a collection of concepts that can be used to describe the structure of a database[1]. Each description generated is a *database schema*. For example, in one database schema there may be a concept 'EMP♯' and in another one there exists the concept 'EmpNr'. If exactly the same thing is meant by these two concepts (e.g. employee number), then we say that there is only a syntactical difference; it can be resolved by syntactical rewriting of the schema. *The semantic level* encompasses the intended meaning of the concepts in a schema. Here, the meaning of a concept depends mostly on its name, which can be interpreted differently by different peo-

[1] *The structure of a database* means the concepts, data types, relationships, and constraints that should hold on the data.

ple. For example, an engineer might consider a peace of metal as a component in an automobile assembly line, while a child would see it as a toy. The challenge is to express the schemas in such a way that the definition of a term is unambiguous and expressed in a formal and explicit way, linking human understanding with machine understanding in an integrated environment.

In this chapter, we will present the principles and general approaches to data integration (cf. Section 2.1). We argue the benefits of conceptual modeling and reasoning in such process in Section 2.2. Consequently, we study different conceptual modeling methodologies for databases in Section 2.3. In this section, we will discuss their features with regard to an explicit and formal expression mentioned above, showing the most suitable conceptual modeling methodology to our objectives.

## 2.1 Information Integration and Data Integration

Information integration provides an integrated view of data stored in multiple sources of probably heterogeneous information. The integration system can be in principle employed to access the data and to update the stored information. The execution of updates on integrated data requires changes in the data sources. Therefore a tight coordination between sources and the integration system and among different sources is needed. This form of integration is typical of interest to Federated database [117, 11].

Recently, a looser approach to integration has emerged, where the autonomy of sources is a basic condition, and the integration system is seen as a data sources' "client" who cannot interfere the operation of its sources. Therefore, the execution of updates on integrated data is not concerned. For this reason, this type of integration is called *read-only information integration*. It is often known under the name of "*data integration*".

In a data integration system, the organization responsible for the system is generally distinct from and independent of thoses controlling individual sources. The user does not access such data sources directly, but poses his or her questions to the integration system, and is thus free from the need to know where the actual data reside and how to access the data sources to extract them. It is the task of the integration system to decide what sources are appropriate to answer the question of the user, to distribute the question on such sources, to collect the responses returned and to present the overall response to the user.

### 2.1.1 Data Integration based on Peer-to-Peer Architecture

Peer-to-Peer (P2P) integration is a possible approach to data integration [63, 134, 56]. This technique consists in building the mappings that implement the connections between data sources (i.e. the peers). Requests are posed to one peer, and the role of the query processing is to exploit, through the mappings, the data that is internal to the peer and even external residing in other peers in the integration system. These mappings allow a database to query another database so that it understands the request. Peers are autonomous systems and mappings are dynamically created and changed. Figure 2.1 presents the general architecture of a Peer-to-Peer data integration.

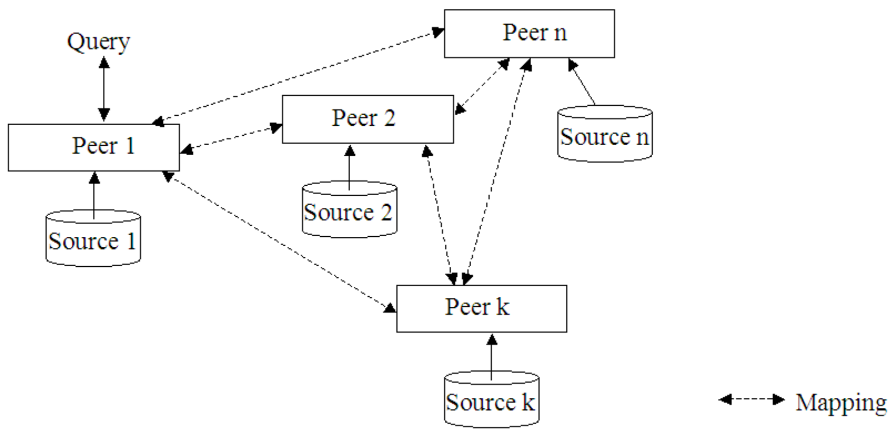


Figure 2.1: Peer-to-Peer data integration architecture

The advantage of this architecture is that it provides an environment easily extensible and decentralized to share data. Any user can contribute new data or even mappings with other peers. However, the disadvantage of this architecture is the combinatorial explosion of the number of connections to be put in place if the number of databases to integrate increases. This raises many challenges. For example, in systems where the peers have the schemas decidable but arbitrarily connected, the indecidability of the query answering may occur [63]. In addition, interconnections between these peers are not under the control of any peer in the system. This is also a challenge in answering queries raised at a peer that takes into account mappings because answers depend heavily on the semantics of the overall system.

## 2.1.2 Data Integration based on Mediated Schema

Most data integration systems employ a *mediated schema*, which is also known as a *global schema*. This is an approach where the entry of a data integration system is a set of data sources that are represented by their schemas (also known as *local schemas*). The integrated view is a global unification of the data coming from the multiple and heterogeneous sources in the system. This view, or uniform query interface, is built on the *mediated schema*. The mediated schema is purely logical and *virtual* in the sense that it is used to process queries but not to store data. The data remain in their data sources. To link the contents of data sources to the elements of the mediated schema, the data integration system employs a set of translation rules called schema mappings. The purpose of mappings is to capture structural as well as terminological connections between the local and the mediated schemas. Given a query from a user, the system employs a set of semantic connections between the mediated schema and local schemas of data sources to translate the user query to queries on data sources, then combine and return the results to the user. Figure 2.2 shows the general architecture of a data integration system based on mediated schema.

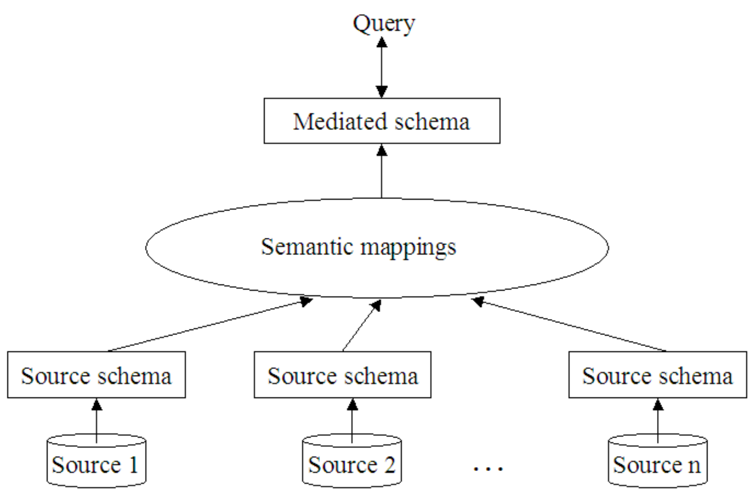


Figure 2.2: Architecture of a data integration system based on mediated schema

While data integration based on mediated schema supports expressive queries on a wide range of autonomous and heterogeneous data sources by exploiting the semantic relationships between different source schemas, some challenges of this approach are:

- Design of a mediated schema must be done carefully and globally;
- Data sources cannot change considerably, otherwise they could violate the mappings to the mediated schema;
- Conceptual notions can only be added to the mediated schema by the central administrator.

### 2.1.3 Procedural and Declarative Approach

There are two principle approaches to data integration, namely the *procedural* and the *declarative* approach.

In the procedural approach, data is integrated in a ad-hoc manner according to a set of predefined requirements of information. In this case, the fundamental problem is how to design software modules so that they access to appropriate sources to fulfil the predefined requirements of information. Some data integration projects following this approach are TSIMMIS [55] and Squirrel [33, 55, 89]. They do not require an explicit notion of integrated schemas, and are based on two types of software components:

- *wrappers* wrap sources;
- *mediators* obtain information from one or more wrappers or other mediators, refine this information through integration and conflict resolution of information coming from different sources, and provides information output to the user or to other mediators.

The basic idea is to have a mediator for all types of query required by the user. Generally, there is no constraint on the consistency of the results retrieved from different mediators. For every two data sources to be integrated, a manuscript or a transformation program should be written and maintained. The latter is the toughest problem, because if a schema changes, the change must be detected and all the transformation manuscripts taking account of this schema, its source and its target must be updated. This requires a lot of maintenance work. Moreover, it is easy to forget several changes if there are so many of them. This problem becomes even harder when many such transformations exist in an enterprise. Therefore, this type of ad-hoc integration is not scalable.

The major part of the current work on data integration adopts the declarative approach for the problem. The goal of this approach is:

- to model the data in a source by an appropriate language;
- to build a unified representation of data;
- to refer to such a representation in querying the integration system; and
- to derive query answers through the appropriate mechanisms accessing sources.

Therefore, this approach assumes that a system of data integration is characterized by explicitly giving users a virtual, reconciled, and unified view of data in a knowledge representation formalism. The latter is often called the *common data model* of the system. The virtual concepts in the view are mapped to the concrete data sources, where the actual data reside, by explicit mapping assertions. Thus, the user formulates his or her question in terms of the common data model, and the system decides how to exploit the mappings to reformulate the question in terms of the language appropriate to data sources.

The declarative approach provides a crucial advantage over the procedural one:

- Although building a unified representation may be costly, it allows us to maintain a consistent unified view of the information sources;
- This view represents a reusable component of a data integration system;
- By this approach, data integration systems can be built by using conceptual modeling and reasoning techniques whose advantages will be shown in the next section.

## 2.2 Conceptual Modeling and Reasoning Advantages

Conceptual modeling (CM) deals with the question on how to describe in a declarative and reusable way the domain information of an application, its relevant vocabulary, and how to constrain the use of data, by understanding what can be drawn from it. CM provides the best vehicle for a common understanding among partners with different technical and application backgrounds. It facilitates information exchange over the Internet and within heterogeneous data sources. Actually, CM of an application domain and reasoning support over conceptual models have become critical for the design and maintenance of Semantic Web services requiring data integration.

The role and importance of CM in traditional architectures for information management systems is well known and tools for CM are commonly used to drive system design [88]. We can give a list of advantages of CM regarding the design and the operation of an integration system as follows.

- *Declarative and system independent approach.* In general terms, one can say that CM is the obvious mean for pursuing a declarative approach to Data Integration. As a consequence, all the advantages deriving from making various aspects of the system explicit are obtained. The CM provides a system independent specification of the relationships within sources.

- *High level of semantics in user interface.* One of the most tangible effects of CM has been to break the barrier between user and system by providing a higher level of semantics in the design and user interface as well. Conceptual models are often expressed in graphic form and even expressed in closed-natural language. This is the key factor in presenting the overall information scenario in user interfaces.

- *Incremental approach.* With CM, the overall design can be regarded as an incremental process of understanding and representing the relationships among data in sources. One can therefore incrementally add new sources into the integration system.

- *Mappings.* While in the procedural approach, the information about the relationships among sources is hard-wired in the mediators, in a declarative approach it can be made explicit. The importance of this clearly emerges when information about data is widespread in separate sources that are often difficult to access and not necessarily conforming to common standards. CM for Data Integration can thus provide a common ground for the overall relationships and can be seen as a formal specification explicit for mediator design. By making the representation explicit, we gain the re-usability of the acquired knowledge, which is not achieved within a procedural approach. This explicit representation facilitates the mappings that can be done not only manually from experts because CM language is understandable for humans, but also semi-automatically because experts can verify them.

Another set of advantages that one can obtain from the introduction of CM into data integration is related to the ability of reasoning over conceptual models. For example, one can use reasoning to check a conceptual representation (i.e. a model) for inconsistencies and redundancies; to maintain the system in response to changes in the information needs; to improve the overall performance of the system, etc. Essentially, if we take an Artificial Intelligence (AI) point of view, we can consider a whole integration system, constituted by an integrated view (with constraints), data sources, and mappings, as a *knowledge base*. In such a knowledge base, knowledge about specific data items (i.e. extensional knowledge) and knowledge about how the information of interest is organized (i.e. intensional knowledge) are clearly separated: extensional knowledge is constituted by the data sources, while intensional knowledge is formed by the integrated view and the mappings. Under this view, computing certain answers essentially corresponds to some reasoning activities or in other words, to some logical inference. The certain answers are those data that are logically implied by the data present in the sources and the information on the view and mapping.

All the advantages outlined above can be obtained by having the appropriate modeling methodologies and modeling languages for representing conceptual model.

A number of conceptual modeling languages have emerged as de-facto standard today, both in Database and Semantic Web technologies, such as ER, UML, ORM for the structured data models (e.g. relational database schemas), XML, RDF(S), DAML+OIL and OWL for the web semi-structured data model. However, many such languages do not have formal semantics based on logic, or reasoners built upon them to support the designer as well as the user. In Section 2.3, we will study some database conceptual modeling which are most popular today. Semantic Web technologies will be introduced in Chapter 3.

## 2.3 Conceptual Data Modeling

Given that most data sources are databases, specifically relational databases, we focus in this section on different conceptual *data* modelings, which are at the basis of the data integration framework. *Conceptual data models* describe the structure of the whole database for a community of users. They were developed to provide a high-level abstraction for modeling data, allowing database designers to think of data in ways that correlate more directly to how data arise in the world. To capture the meaning of an application domain as perceived by its developers, conceptual models hide the physical storage structures and provide models that are more understandable to human. Database schemas could first be designed in a high-level semantic model and then translated into one of the traditional models for ultimate implementation. Capturing more semantics at the schema level allows the designer to declaratively represent relevant knowledge about the application. It follows that sophisticated types of constraints can be asserted in the schema, rather than embedding them in methods. Besides, it implies more possibilities to reason over the content of the database. Such reasoning can be exploited for deriving useful information for the solution of new problems posed by data integration (e.g. schema comparison and integration [35]). Thus, integrating semantics of data sources should be from conceptual models of these sources rather than from their *logical* or *physical models.* Before discussing the most popular conceptual data models, let us have a look at these three model levels in relational database design.

### 2.3.1 Categories of Data Models

The first step in meeting the communication challenge is to recognize that each database appears in different views, or data models. Each model is important and serves its own purpose, but not all are useful or even meaningful to everyone who participates in the design process.

**The physical model**. It is a view that is restricted to a specific relational database and to the physical implementation of that database. At this level, implementation details such as data types (e.g. "VARCHAR(5)") or creation syntax (e.g. vendor specific SQL dialects) are shown. In more simple terms, a physical model is the code of the Data Definition Language (DDL, often, in SQL). In this language, many types of constraints can be defined, either via columns of table, or using more complex techniques such as the use of assertions or triggers. However, for the user and performance requirement, physical models usually contain redundant data or information. For example, in physical models performance and convenience factors may cause new or altered indexes need to be created to speed up the query time; new other interfaces (e.g. views, stored procedures or additional methods) may be added. Figure 2.3 is an example of a simple database schema described in a physical model. Since end users, clients, and most project managers are usually not steeped in such typical database terminology such as clustered indexes and foreign keys, the physical model is not an effective means of communication outside the realm of designers and programmers.

```
CREATE TABLE  Dept_Mgr(
   dno      INTEGER,
   ssn      CHAR(11) NOT NULL,
   fromDate   DATE,
   toDate  DATE,
   PRIMARY KEY  (ssn, fromDate),
   FOREIGN KEY  (dno) REFERENCES Departement,
   FOREIGN KEY  (ssn) REFERENCES Employes,
      ON DELETE NO ACTION);
```

Figure 2.3: A simple schema in physical model

**The logical model**. Logical models serve as the basis for the creation of physical models. Logical models are biased towards the solution of the problem (designing). They need to specify technical details and so often use non-natural or machine language. However, logical models may not always correspond to physical models. In logical level, normalization [47] occurs while denormalization may occur in physical level because of user or performance requirements. Therefore, logical model is often transformed into physical model by applying several simple modifications.

The logical model is portrayed in the tabular form of *relational model* and shows the logical relationships between tables. The relational model [47] represents the database as a collection of tables of values, where each row represents a collection of related data values. The table name and column names are used to help in interpreting the meaning of the values in each row. All values in a column are of the same data type. In the formal relational model terminology, a row is called a *tuple*, a column header is called an *attribute*, a table is called a *relation*. Each relation is given a subset of attributes which is called a *primary key* to uniquely identify each tuple. A *relation schema* is made up of the relation name and a list of attributes. A *relational database schema* is a set of relation schemas and a set of integrity constraints such as the primary key constraint, the foreign key constraint, etc. The *foreign key* states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. The foreign key is used to maintain the consistency among tuples of the two relations. Figure 2.4 shows an example of a relational database schema, where the underlined attributes make up the primary key; the attribute deptNo of Location_Dept is a foreign key that gives the department number for its location and thus its value in every Location_Dept tuple must match the id value of some tuple in the Department relation.

The limitation of relational models is that the meaning of a relation is entirely defined by the set of tuples that correspond to the relation. This means the meaning of a relation is not self-contained. The relation name and attribute names have no formal meaning. In the relational model, there are no explicit relationships between different relations. These relationships in relational database schemas must be retrieved using queries on the database and some additional knowledge (e.g. a set of integrity constraints) that is not contained in the relational model.

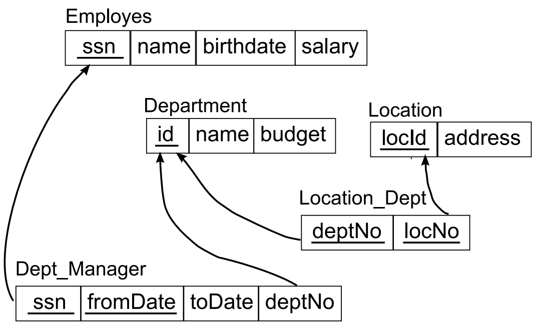


Figure 2.4: An example of the database schema in the logical model

**The conceptual model**. Conceptual modeling provides richer data structuring capabilities for DB applications. A modeling method comprises a language and also a procedure for using the language to construct models. The language may be graphic (i.e. diagrams) and/or textual. In contrast to logical and physical models which specify underlying database structures to be used for implementation, conceptual models portray applications by using terms and concepts familiar to the application users. Conceptual models are then (automatically) translated (by the tool) into the corresponding logical schemas for the target Database Management System (DBMS).

We can use the following criteria [66] as a useful basis for evaluating conceptual modeling methods: expressibility, clarity, semantic stability, validation mechanisms and formal foundation. The *expressibility* of a language is a measure of what it can be used to say. Ideally, a conceptual language should be able to model all conceptually relevant details about the application domain. The *clarity* of a language is a measure of how easy it is to understand and use. *Semantic stability* is a measure of how well models or queries expressed in the language retain their original intent in the face of changes to the application. The more changes one is forced to make to a model or query to cope with an application change, the less stable it is. *Validation mechanisms* are ways in which domain experts can check whether the model matches the application. A *formal foundation* ensures models are unambiguous and executable (e.g. to automate the storage, verification, transformation and simulation of models).

Several modeling methods have been proposed. The most popular ones are Entity-

Relationship, Unified Modeling Language and Object Role Modeling. We will investigate these mechanisms with regard to the above criteria in the next sections.

### 2.3.2 Entity-Relationship Model

One of the most widespread conceptual models becoming a standard and extensively used in the design phase of applications is the Entity-Relationship (ER) model [37]. In the ER model, the domain of interest is modeled by means of an ER schema. The basic elements of the ER schema are *entities*, *relationships*, and *attributes*. An *entity* denotes a set of objects that have common properties. Such objects are called the *instances* of the entity. Elementary properties are modeled through *attributes*, whose values belong to a predefined domain such as String, Boolean. For example, an EMPLOYES entity may be described by its attributes: EMPLOYES's name, birthdate, salary (cf. Figure 2.5). Properties of that are due to relations to other entities are modeled through the *participation* (total or partial) of their entity in relationships. A *relationship* denotes a set of tuples each of which represents a combination of instances of the entities that participate in the relationship. Since each entity can participate in a relationship more than once, the notion of *ER-role* is introduced, which represents such a participation and to which a unique name is assigned. The *arity* of a relationship is the number of its ER-roles. *Cardinality constraints* can be attached to an ER-role in order to restrict the number of times each instance of an entity is allowed to participate via that ER-role. Such constraints can be used to specify both existence dependencies and functionality of relations. They are often used in a restricted form, where the minimum cardinality is either 0 or 1 and the maximum cardinality is either 1 or $\infty$. Additionally, so called $is - a$ relations are used to represent inclusion assertions between entities, and therefore the inheritance of properties from a more general entity to a more specific one. An important constraint is the *key* constraint on attributes. The attributes whose values are distinct for entity instances are called key attributes of that entity. For example, the social security number (ssn) attribute is a key of the EMPLOYES entity because no two employees are allowed to have the same ssn. The ER model uses diagram notations to describe ER schemas (Figure 2.5 is an example). In the ER diagram, the key attributes are indicated by the underlines.

The ER model is common in use because of the correspondence with the relational model, and the known translations from ER diagrams to relational models [47]. However, while in ER modeling, entities are explicitly specified and have a meaning, they only survive as meaningless names in the relational model (i.e. relation names). Furthermore, the relationships explicitly expressed in the ER model, are

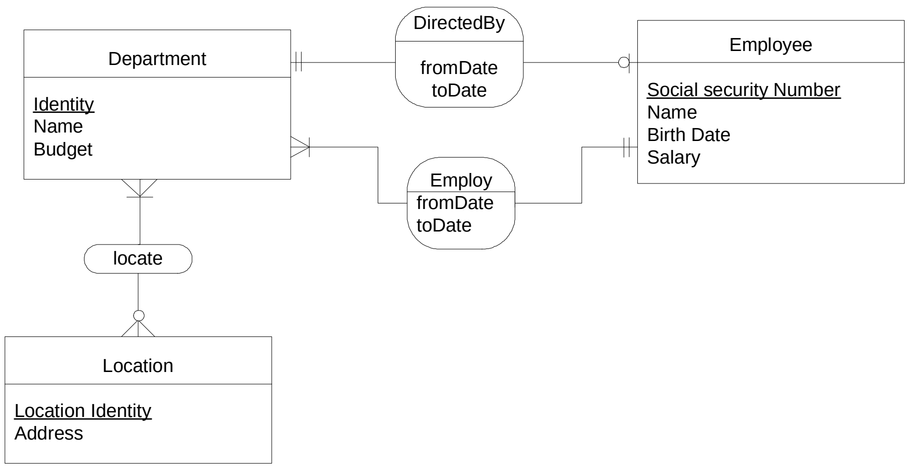


Figure 2.5: An example of the ER schema

hidden in the relational model.

Even though the ER model seems relatively simple (in the simple case just consisting of entities and relations), ER models can become quite complex, even for relatively simple domains [128]. Besides, ER modeling lacks the formal and real-world semantics. Reasoning on the ER schema has therefore been objectives of several works [31].

### 2.3.3 UML Class Diagram

The Unified Modeling Language (UML) [104] has become widely used for software and database modeling. UML data modeling provides more expressive power in database schema definition. Indeed, several modeling constructs of UML data models are borrowed from the research on semantic data modeling and semantic networks in Artificial Intelligence (AI). For modeling notation, UML uses class diagrams to which constraints in a textual language may be added. Essentially, class diagrams provide an extended ER notation. Figure 2.6 shows how the ER schema of Figure 2.5 can be displayed using UML notation.

UML class diagrams allow for modeling, in a declarative way, the static structure of an application domain, in terms of classes and relations between them. A *class* denotes a set of objects with common features, like entity in ER modeling. A class is graphicly rendered as a rectangle divided into three parts (see, e.g., Figure 2.6). The first part contains the *name* of the class, which has to be unique

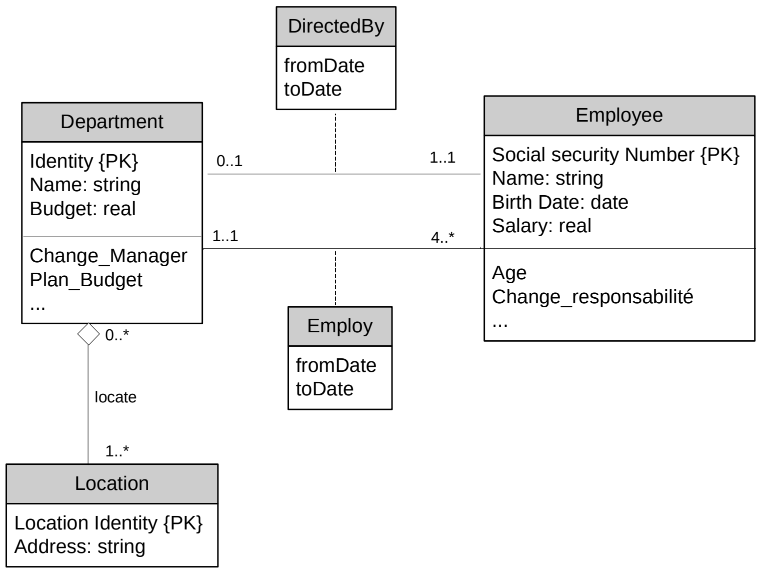


Figure 2.6: UML diagram of the example in Figure 2.5

in the whole diagram. Two other optional parts contain the *attributes*, like ER modeling, and the operations of the class, i.e. the operations associated to the objects of the class. An attribute *a* of type T for a class C associates to each instance of C a set of instances of T. Attributes are unique within a class, but two classes may have two attributes with the same name, possibly of different types. An *association* in UML is a relation between the instances of two or more classes. Therefore, associations can be bi-, thre- or n-ary. An *association class* describes the properties of an association, such as attributes, operations, etc. A particular kind of binary associations are *aggregations*, which denotes a part-whole relationship, i.e., a relationship that specifies that each instance of a *parent* class (the containing class) contains a set of instances of a *child* class (the contained class). In UML one can use a *generalization* between a parent class and a child class to specify that each instance of the child class is also an instance of the parent class (like is-a relations in ER modeling). Hence, the instances of the child class inherit the properties of the parent class, and satisfy additional properties that in general do not hold for the parent class. Several generalizations can be grouped together to form a class hierarchy.

UML allows cardinality constraints on both attributes and associations. Disjointness and covering constraints are in practice the most commonly used constraints in UML class diagrams. Nevertheless, some useful representation mechanisms are not considered in UML schemas such as subsetting of attributes, inverse of attributes, union and complement of classes, mainly because it is commonly accepted that part of the semantics of the application can be represented within methods. UML allows for other forms of constraints such as specifying class identifiers, functional dependencies for associations, etc., generally through the use of OCL (Object Constraint Language) [113], a form of constraint expressible in First-order logic (FOL) [52]. However, unrestricted use of OCL constraints makes reasoning on a class diagram undecidable, since it amounts to full FOL reasoning. Besides, the use of UML in industrial-scale software applications brings about class diagrams that are large and complex to design, analyze, and maintain. The expressiveness of the UML constructs may lead to implicit consequences that can go undetected by the designer in complex diagrams, and cause various forms of inconsistencies or redundancies in the diagram itself. This may result in a degradation of the quality of the design and/or increased development times and costs. Several works [15, 53, 49, 70] have proposed to describe UML class diagrams using various formal kinds in order to reason on UML class diagrams, and formally prove properties of interest through inference, and hence help the designer in understanding the hidden implications of his choices when building a class diagram.

### 2.3.4 Object Role Modeling

Object-role modeling (ORM) [68] is a conceptual modeling method for designing and querying data models. Typically, a modeler develops a data model by interacting with others who are collectively familiar with the application. These subject matter experts need not have technical modeling skills. Therefore, reliable communication occurs by discussing the application at a conceptual level, using natural language and analyzing the information in simple units. ORM is specifically designed to improve this kind of communication.

Originally, ORM is a successor of NIAM (Natural language Information Analysis Method) [139] that was developed in the early 70's to be a stepwise methodology arriving at "semantics" of a business application's data based on natural language communication. Relational terms such as tables, columns, and keys do not exist in ORM because they are abstractions used for describing things stored in a relational database. Actually, Object Role Modeling got its name because it views the application world as a set of objects (*entities* or *values*) that plays *roles* (parts in relationships). ORM sometimes can be called as fact-based modeling because it verbalizes the relevant data as *elementary facts*. These facts cannot be split into smaller facts without losing information. The terms used in ORM have a direct relevance to concepts in the real world. Thus, this view of the world lets you describe the data in everyday terms rather than using an artificial language that is not effective in communicating and prone to being misunderstood. *Value object types* (or simply Value types) and *entity object types* (or simply Entity types) are distinguished by linguistic distinction. Value types correspond to utterable entities (e.g. 'Code', 'Firstname') while entity types refer to "non-utterable" entities (e.g. 'City', 'Person'). The relations between objects generates facts. ORM allows relationships with one role (for example, Person works), two, three, or as many roles as you like.

ORM has a rich language for expressing business rules, either graphicly or textually (i.e., the graphic form can be translated into pseudo natural language statements). In an ORM diagram, roles appear as boxes, connected by a line to their object type. A *predicate* is a named, contiguous sequence of role boxes. Since these boxes are set out in a line, fact types may be conveniently populated with tables holding multiple fact instances, one column for each role. This allows all fact types and constraints to be validated by verbalization as well as sample populations. Communication between modeler and domain expert takes place in a familiar language, backed up by population checks. It is easy to express virtually any style of static constraints upon the ORM diagram, such as mandatory, uniqueness (i.e. cardinality), subset, equality, exclusion, etc. These constraint types may also be combined. For example, an exclusion constraint may be combined with mandatory disjunction to specify that an object must either play role $x$ or role $y$ but not both.

Suppose Table 2.1 includes data about departments employing employees. For simplicity, we assume that employees are identified by their names. The first row contains three elementary facts: the department identified "IT" employs the employee named "Hang DO", the department identified "IT" employs from year "1996" and the department identified "IT" employs to year "2001". The null value "?" indicates the absence of a fact to record a department MK's end date. All the facts are elementary rather than compound. Therefore, null values do not appear in verbalization and eight facts can be extracted from Table 2.1.

Table 2.1: Some data of the department employment

| Department | Employee | fromDate | toDate |
|---|---|---|---|
| IT | Hang DO | 1996 | 2001 |
| MK | Marie CLAIRE | 2000 | ? |
| MK | Hang DO | 2002 | 2005 |

Although Table 2.1 includes eight fact instances, it has only three fact types: Department employs Employee, Department employs from Date and Department employs to Date. The facts in this table and the constraints on them can be modeled in ORM as in Figure 2.7. In this figure, we can see the population of the facts corresponding to each fact type. (Note that the population is not part of the conceptual schema itself.)

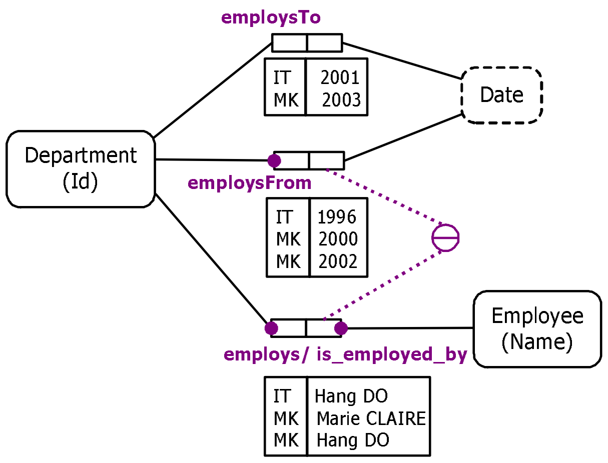


Figure 2.7: Modeling the data of Table 2.1 in ORM

ORM has well-defined formal semantics. Actually, it has been fully formalized in FOL [131, 135, 132]. Conceptual query languages have also been designed for ORM like ConQuer [21]. ConQuer supports not only ORM queries but also the automatic translation from ORM to SQL queries. Furthermore, ORM includes a vast array of schema transformations as well as optimization heuristics to determine which transformations to use. For implementation, ORM schemas are usually mapped to relational database schemas, in which many fact types may be grouped into a single table (cf. Chapter 7). Actually, a correctly described ORM schema can be translated into a 5th normal form (5NF) relational database schema [13, 12]. A reverse engineering can also be provided to convert a relational database into an ORM schema [137].

### 2.3.5 Distinguish ORM from ER modeling and UML class diagramming

Unlike ER modeling (or simply ER) and UML class diagramming (or simply UML), ORM treats all elementary facts as relationships, thus regarding decisions for grouping facts into structures (e.g. relation schemas, attribute-based entities, classes) as implementation concerns irrelevant to business semantics. The limits of ER and UML on conceptual data modeling with regard to the ORM method have been investigated in a series of works (e.g. [64]). In this section, we will review the main features that distinguish ORM from ER and UML. Note that UML is an extension of ER. Therefore, besides the limits of the extended part of UML, the limites of ER that we show here are also applied to UML.

**Clarity limit**. The semantics of ER and UML does not correspond to the real-world semantics of the domain of interest, but must accommodate itself to the ER and UML model. ER (UML) often requires to make *arbitrary decisions* about what an entity (class), a relationship (association), or an attribute is. Even experienced modelers cannot think of data in ways that correlate to how data arise in the world. Consequently, the ER (UML) model is incomprehensible for users. Hence, although many users of tools for automating ER (UML) believe that they are creating a conceptual model, referring to the conceptual modeling criteria mentioned above, they are not. Actually, ER and UML can be classified to be at logical abstraction level [14]. To illustrate this, we will use examples that show the differences in the resulting schema in ER (UML) and ORM.

- *Illogical attributes.* Suppose adding to the schema given in Figure 2.7 two fact types Employee lives in City and Employee works in City. In ORM, an Employee and a City are both conceptual objects (entities). You could tell someone that you (an Employee) works in Sophia-Antipolis (a City). But you would never try to explain to someone unfamiliar with database design that a City could ever be an attribute of an Employee. Let us say the attributes of an Employee may be the weight, height; but City could not logically an attribute of a person. Furthermore, City entity is converted to two attributes of the Employee (LiveCity) and (WorkCity) in an ER (UML) schema. There is no such thing as a LiveCity or a WorkCity in the real world. Accordingly, the real-world semantics of the domain of interest is missing in ER (UML) schema. Moreover, it is not relationally possible to have two attributes with the same name, such as City and City, in the same logical entity. You have to create attribute names that combine abbreviated semantic and domain information into a single attribute name. Figure 2.8 shows the two fact types that would be designed in ORM (a) and that would appear in an ER (UML) schema (b).

- *Entities without real-world semantics.* Suppose adding to the simple schema given in Figure 2.8(a) a fact that an Employee has missions in multiple cities. Conceptually, this addition is simple: Add another fact type to the schema, reusing the Employee and City objects and naming the new role played as shown in an ORM schema in Figure 2.9(a). Note that the schema contains *only one* Employee object and *one* City object. But in ER (UML), to add this new information, you must establish a logical entity named MissionCity

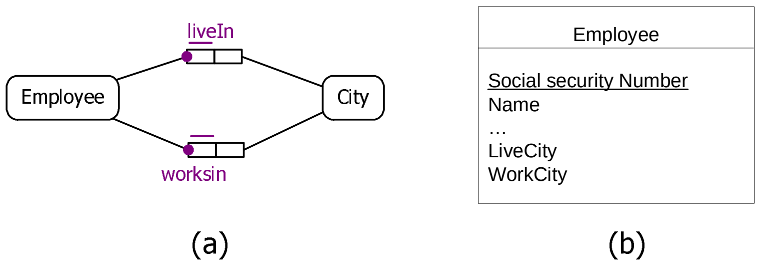


Figure 2.8: An example illustrates the illogical attributes of an object in ER (UML) modeling

(or something similar) and their attributes (ssn of the Employee and City) to store all the Cities in which a given Employee has missions in (cf. Figure 2.9(b)). However, is there such a thing as a MissionCity entity? The answer is that this entity is not conceptual and was arbitrarily determined to accommodate the ER (UML) mechanisms.

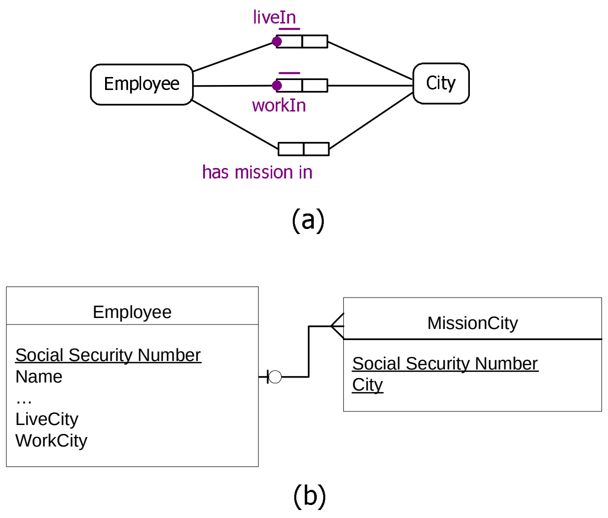


Figure 2.9: An example illustrates the illogical entities in ER (UML) modeling

**Attribute-based modeling limit.** Attribute-based modeling in ER (UML) is also a source of many other problems that ORM does not encounter thanks to its attribute-free feature:

- *Unstable schema.* Suppose we decide to add the fact type: 'City has Monument'. This addition would now force us to show City as an entity type in ER or class in UML, so we would have to replace the attribute description by relationships. This is a significant change to the schema (cf. Figure 2.10(b)). While in ORM, all we have to do is adding a new fact type; nothing else changes; and we gain the added benefit of revealing the conceptual object types (semantic domains) that bind the schema (cf. Figure 2.10(a)).

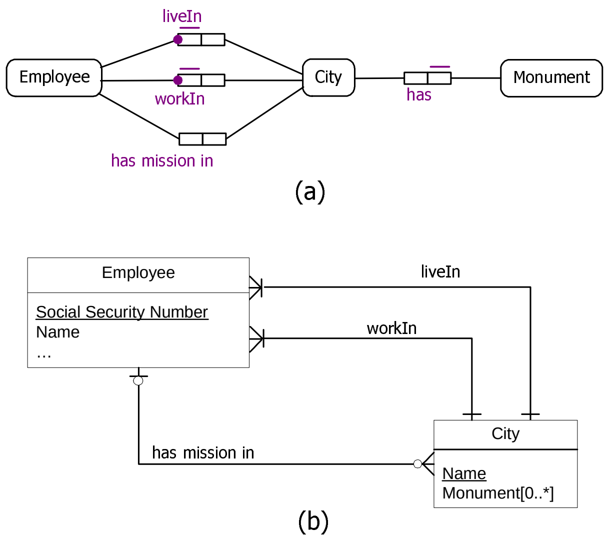


Figure 2.10: An example illustrates the instability of ER (UML) schemas

- *Cumbersome verification.* Attributes make it awkward to talk about fact populations. ER (UML) diagrams are too cumbersome for performing the population checks that are so vital for validating rules with clients.
- *Limited and complex expressivity.* Displaying some facts as attributes and some as relationships leads to the requirement for different notations to express the same kind of constraint or rule. Avoiding attributes in ORM leads to greater simplicity and uniformity. We do not need notations to reformulate constraints on relationships into constraints on attributes or between attributes and relationships. Apart from this unnecessary complexity, some ER/UML notations do not let you express a constraint on an attribute, even that constraint could be expressed if the fact is modeled as a relationship (e.g. subset).

The limites of ER (UML) shown above do not mean that using attributes is not useful. Once having a full schema, displaying less important features as attributes can help provide a compact view of the schema. Actually, ORM includes abstraction techniques so that you can display "minor" fact types as attributes (e.g. using unary predicate). Besides, ORM schema can be converted to ER (UML) diagram and vice versa [137, 22, 121]. Therefore, if you like to obtain an ER (UML) diagram, the best way is abstracting it from an ORM schema.

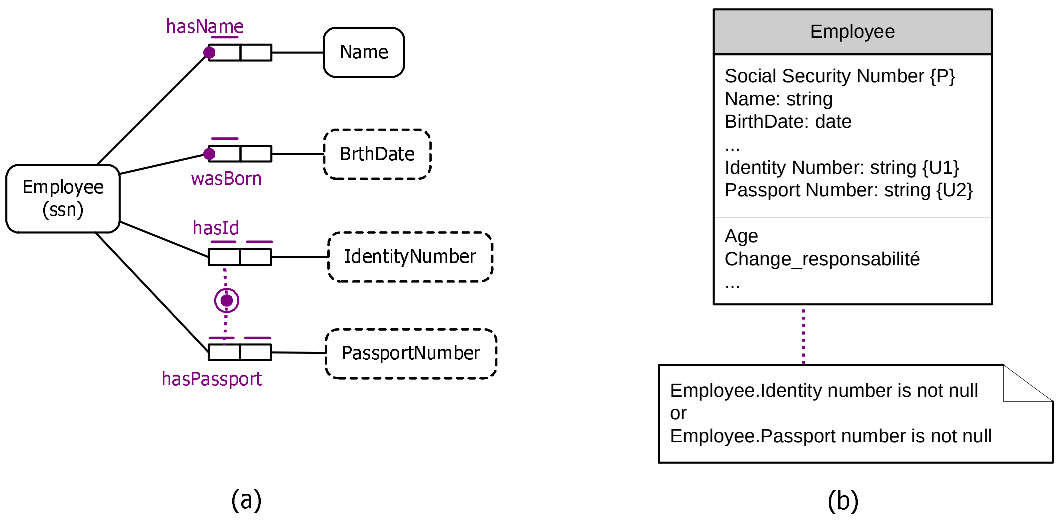


Figure 2.11: Constraintes in ORM (a) are described as an attached comment in UML (b)

**Notation expressivity.** ORM graphic notations are far more expressive than those of ER and UML. They allow constraints to be applied wherever they makes sense. UML does not have a graphic notation for *disjunctive mandatory* roles. For example, we would like to express that an employe must have an identity number or a passport number (see its diagram in ORM and UML in Figure 2.11). As seen at the bottom of Figure 2.11 (b), in UML this kind of constraint needs to be expressed textually in an attached comment. Even such constraint can be expressed in some formal language, e.g. OCL, the readability of the constraint in UML is typically poor compared with the ORM verbalization (e.g. **each** Employee has **a** SocialSecNr **or** has **a** PassportNr).

UML does not have a *standard* graphic notation for cardinality constraints on attributes but use the textual constraints in braces after the attribute names (P = primary identifier, U = unique, with numbers appended if needed to disambiguate cases where the same U constraint might apply to a combination of attributes). Or else a tool extension is needed [3], but clearly this is not portable.

Although UML provides *xor-constraints* between single roles, the standard seems to imply that these roles must belong to different associations. If so, UML cannot use an xor-constraint between roles of a ring fact type (e.g. between the husband and wife roles of a marriage association). ORM exclusion constraints cover this case, as well as many other cases not expressible in UML graphic notation. The pair-exclusion constraint in ORM can be expressed in UML by adding a textual constraint as a comment written in some language (e.g. OCL), and connecting this by dotted lines to the two associations. However this notation is both cluttered and non-standard (since UML allows users to pick their own language to write textual constraints). UML has no graphic notation for exclusion between attributes, or between attributes and associations. In these cases, we must resort to non-standard notations or textual constraints or use alternative ways to model them. ORM's exclusion constraint applies not just to a set of roles, but a set of role-sequences. It is clear that UML's xor-constraint is far less expressive than ORM's exclusion constraint.

ORM allows a *subset constraint* to be graphicly specified between any pair of compatible role-sequences. This constraint is not possible in any variation of ER modeling or in the UML class diagram without introducing intermediate (and unnatural) structures. Actually UML allows subset constraints to be specified between whole associations, but does not provide a graphic notation for subset constraints between single roles or between parts of associations. Instead, UML must use a comment including the textual constraint to express the single-role subset constraint. For example, the constraint that "Employees have second names only if they have first names" can be represented as a subset constraint in ORM. If a UML diagram depicted the relationship "have" as an association, it would not be able to capture the subset constraints. If firstName and secondName are modeled as attributes of Employee, the single-role subset constraint can only be expressed by attaching a comment: Employee.firstName **is not null or** Empployee.secondName **is null**. *Equality constraint* is a shorthand for two subset constraints. As a consequence, UML has no graphic notation for this kind of constraints either. Subset and equality constraints enable various classes of schema transformations to be stated in their most general form. ORM's more general support for these constraints allows more transformations to be easily visualized.

With regard to objectified relationships (association classes), UML requires the same name to be used for the original association and the association class, impeding natural verbalization of at least one of these constructs. In contrast, ORM nesting is based on linguistic normalization (i.e. a verb phrase is objectified by a noun phrase), thus allowing both to be verbalized naturally, with different names for each. ORM's concept of an *external uniqueness constraint* that may be applied to a set of roles in one or more predicates provides a simple, uniform way to capture all of UML's qualified associations and unique attribute combinations, as well as other cases not expressible in UML graphic notation (e.g. cases with m:n predicates or long join paths). UML does not provide *ring constraints* built-in, *join constraints* (even it frequently arises in real applications). Since UML does not provide standard graphic notations for such constraints and leaves it up to the modeler whether such constraints are specified, it is perhaps not surprising that many UML schemas one encounters in practice simply leave such constraints out.

ORM's graphic constraints can be automatically mapped to efficient SQL. By contrast, code generated from UML's OCL constraints is unlikely to be as efficient as hand crafted SQL. UML constraints could be expressed in SQL but at the cost of both being error prone and less clear. As always, the ORM notation has the further advantage of facilitating validation through verbalization and multiple instantiation.

**Conceptual Query**. Apart from conceptual modeling, ORM is ideal for performing queries at the conceptual level. You can query a database without any knowledge of how the facts are grouped into implementation structures. The reason is that when adding new information in ORM model, you only need to extend the schema and nothing else changes (i.e., there is no reformulation in existing schema). Therefore conceptual queries may be formulated in terms of continuous paths through the schema. Moving from a role through an object type to another role amounts to a conceptual join. ER (UML) diagrams, however, typically omit domains, so you must look them up in a table.

Suppose you want to list the titles of the projects that have an assessor. This request may be formulated as the following ORM query: "List the ProjectTitle of each Project that was assessed by an Employee". If a project has at most one assessor, this query generates the following SQL:

```
select projectTitle from Project
where assessor is not null
```

If the application allows more than one assessor per project , you do not need to change the ORM query, because constraints have nothing to do with the meaning of the query. However, the relational structures have changed and the following SQL query is generated:

```
select X1.projectTitle from Project X1, Assessment X2
where X1.projectnr = X2.projectnr
```

An SQL query often needs to be changed if the relevant part of the conceptual schema or internal schema is changed (even if the meaning of the query is unaltered). However, more substantial queries in SQL than the example above can be formulated easily as conditioned paths through ORM space.

ORM scores higher on clarity, because its structures may be directly verbalized as sentences, it is based on fewer and more orthogonal constructs, and it reveals semantic connections across domains. Being attribute-free, ORM is more stable for both modeling and queries. In addition, ORM is easier to validate. In ORM all facts are expressed in the same way, using roles. As a result, the notation is both uniform and simple to populate.

## 2.4 Conclusion

In this chapter, we have introduced the principle concept of integrating information in general and data in particular. We also showed the advantages of declarative approach and of conceptual modeling and reasoning in designing a data integration system. It turns out that data integration tools typically require a common and comprehensive schema design before they can be used to share information. But with attributed-based models, these tools are difficult to extend because schema evolution is heavyweight and may break backward compatibility. ORM, however, is capable of being evolved without breaking the backward compatibility.

By following the fact-oriented approach and avoiding attributes, ORM avoids arbitrary modeling decisions and highlights relationships through semantic domains, enhances semantic stability, and facilitates natural verbalization. This stability applies not only to the conceptual schema itself, but also to conceptual queries based on the schema. For data modeling, fact-oriented graphic notations are far more expressive, clearer and less error prone than ER and UML graphic notations. Actually, semantics in ORM are expressed in a formal way thanks to its underlying logic language FOL.

Since ORM includes procedures for mapping to attribute-based structures. We believe that the favorite solution is using ORM first to do the conceptual modeling, getting all the benefits of its simplicity and richness, and then applying to it mapping procedures to generate other views (such as Relational model, ER, UML).

Moreover, if ORM schemas are represented explicitly, there will be a significant scope for inter-operability. Nowadays, several tools have been developed to convert graphic ORM schemas to those in XML format [43, 120]. Even though the description of ORM schemas in XML by these tools is still in syntactic level, it will assist inter-operation tools to exchange or parse ORM schemas.

Nevertheless, semantics of an application domain represented in conceptual data models are now typically in diagram formats, and used in an off-time mode, i.e. used during the design phase, not at run-time of applications. While in the Web environment, applications and application types in general are unknown a priori, including the manner in which they will want to refer to the data, or more precisely, to the concepts and attributes that take their values from the database. Maintenance in the systems directly integrating database schemas would become nearly impossible. Therefore, elements of meaning (or knowledge) of the database's underlying domain have to be agreed and represented explicitly. They need to be stored, accessed, and maintained externally to the database schema as well as to the intended applications. This is where *ontologies* come into play.

In the next chapter, we will study ontology technologies on the Semantic Web and how to integrate relational data sources into this environment.

CHAPTER 3

# Ontology Technology and Data Integration into Semantic Web

In the previous chapter, we have examined the problem of data integration, the role of conceptual modeling in data integration and conceptual modeling for relational data sources. In this chapter, we will investigate the environment into which relational data sources should be integrated (Section 3.1). Based on the argument in Section 2.4, we focus on how to represent knowledge base in the Semantic Web. This leads us to study the Web Ontology technology, which includes not only the formalisms for knowledge representation on the Web but also their underlying mechanisms to store the semantics and to reason on it. We concentrate on OWL in Section 3.2, the current standard Web Ontology language proposed by W3C[1]. Then in Section 3.3, we introduce the underlying logic foundation of OWL, which guarantees not only its formal semantics but also its decidability. After that, in Section 3.4 we will investigate the recent research on data integration from relational data sources into the Semantic Web environment, focusing on how to represent the semantics of data in relational data sources on the Semantic Web. In consequence, we propose our approach of integrating relational data sources into this environment in Section 3.5.

[1]World Wide Web Consortium. http://www.w3.org/

## 3.1 Semantic Web Environment

Being an effort of a large number of researchers and industrial partners led by W3C, the Semantic Web (SW) is an extension of the current Web [18]. Like Internet, the SW is decentralized as much as possible. It provides a common framework that allows to share and reuse data through community frontiers and applications. In this environment, the resources such as documents, images, real world objects like persons or institutions, or even services have the associated meaning (i.e., the information has the well-defined semantics); these resources can be readily accessible to automated processes; and the interaction capacity between human and machine is improved. In other words, the SW is developed to represent the information which is understandable to humans (i.e. knowledge) and tractable to machines. The SW can be seen as an effective infrastructure to improve the visibility of knowledge on the Web. Ontologies in the SW environment, as introduced in Section 3.1.2, are supposed to hold application-independent domain knowledge while conceptual schemas were developed only for the use of enterprise applications. Ontologies are intended to represent agreed and shared domain semantics. As a result, with ontology technology, the SW provides an environment in which heterogeneous information resources can be integrated; computer systems can meaningfully communicate to exchange data and make transactions interoperate independently of their internal technologies.

### 3.1.1 Semantic Web Architecture

The architecture of the SW is based on a hierarchy of languages. Each language both exploits the features and extends the capabilities of the layers below (cf. Figure 3.1[2]). Therefore, this architecture is also called "Semantic Web Language Layer".

Unicode-URI and XML (Schema) consist of existing standards for data representation and provide a syntactical basis for the Semantic Web languages.

*Unicode* provides an elementary character encoding scheme, which is used by XML. *Uniform Resource Identifier* (URI) is a fundamental component of the current Web, which provides the ability to uniquely identify resources as well as relations among resources on the Web. URI is the generic set of all names/addresses that

[2]The figure is motivated by the architecture proposed by Berners-Lee [17]

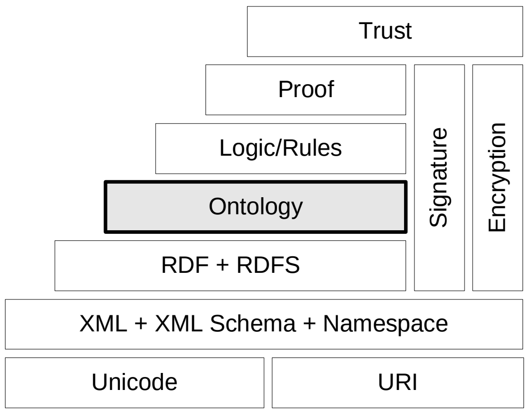


Figure 3.1: The basic layers of data representation for the Semantic Web

are short strings refering to resources. All concepts used in the languages in the upper layers are specified using Unicode and are uniquely identified by URIs.

*eXtensible Markup Language* (XML) [127] is a fundamental component for syntactical interoperability. XML is the universal format for (semi-)structured documents and data on the Web, proposed by the W3C. The main contribution of XML is that it provides a common and communicable syntax for web documents. Data is described using a number of tags with arbitrary names that can contain other tags or arbitrary data. Tags are used to build the elements of an XML document, which are identified by name and may have a set of attributes. Each attribute specification has a name and a value. XML allows users to define their own tags. It can be used as a data exchange format, in which case all parties taking part in the exchange need to agree on a common structure for the XML document. Another possible use of XML, as it is done in the Semantic Web, is to use it as the serialization[3] language for other languages.

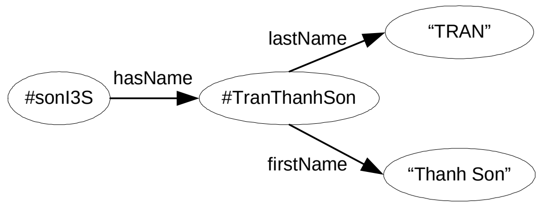
The structure of XML documents is prescribed by *XML Schema*. It contains a set of rules (constraints on the structure and on the content of the XML document) with which an XML document must comply to be considered as valid according to the schema. XML Schema describes only the physical structure (i.e. the syntax) of an XML document but not the meaning (i.e. the semantics) of the data. XML Schema has a broad range of (simple) data types (e.g. integer, string), and the

[3]Serialization is the process of converting an object into a form that can be readily transported or persisted to a storage location.

possibility to form (arbitrarily) complex data types (e.g., positive integers make up a sub-type of integer).

*Namespace* is a context or an abstract container of names, terms, and words that represent objects and concepts in the real world. A defined name in a namespace corresponds to only one object.

*Resource Description Framework* (RDF) [99] was developed by the W3C as part of its semantic web effort. RDF is a model for representing metadata about Web resources. This information is identified on the Web by the URIs and needs to be processed by applications, rather than being only displayed to people. Resources may be divided into groups called *classes*. RDF describes metadata by using RDF statements, which are ¡subject, predicate, object¿ triples. The *subject* is a resource and can be the object of another statement. The *predicate* is a property describing the given resource and expresses a relation between the subject and the object. The *object* is the object of the relation and can be a value or an RDF statement assigned to property. An RDF model can be represented as a directed graph, where the subjects and objects form the nodes and the predicates form the arcs. RDF also provides an XML-based syntax (called RDF/XML) for recording and exchanging the metadata. An example of RDF graph, the corresponding triple notation and the corresponding RDF/XML serialization are shown in Figure 3.2.



(a) RDF Graph

```
(hasName, #sonI3S, #TranThanhSon)
(lastName, #TranThanhSon, "TRAN")
(firstName, #TranThanhSon, "Thanh Son")
```

(b) RDF Triples

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:s="http://SonClub.org/schema/">
<rdf:Description about="sonI3S">
<s:hasName>
<rdf:Description about="#TranThanhSon">
<s:lastName>TRAN</s:lastName>
<s:firstName>Thanh Son</s:firstName>
</rdf:Description>
</s:hasName>
</rdf:Description>
</rdf:RDF>
```

(c) RDF/XML format

Figure 3.2: An example of RDF graph, its triples and RDF/XML serialization

RDF gives a formalism for metadata annotation, and a way to write it down in XML, but it does not give any special meaning to vocabulary such as a subclass of another class or a type of a given class. Therefore, an interpretation in RDF is an arbitrary binary relation.

*RDF Schema* (RDFS) [25] is a language to describe vocabularies, classes and the properties that may be used to describe classes, properties and other resources in the RDF model. RDFS is a semantic extension of RDF. It further extends RDF by adding more modeling primitives[4] commonly found in ontology languages like domain and range restriction on property, class and property taxonomy[5], etc. RDFS allows to define vocabulary terms and the relations between those terms. The combination of RDFS and RDF is usually denoted as RDF(S).

Ontology layer is intended to bring formal languages to the reasoning. It plays an important role in fulfilling semantic interoperability and is considered as the core of the SW. Together with the Logic layer that enable the writing of rules, Ontology layer provides the capability to deduce new knowledge (cf. Section 3.3). We will examine Ontology in more detail in the next sections. The proof layer, based on the use of rules, evaluates, together with the trust layer, the mechanism for applications to decide whether to trust the given proof or not. The Proof and Trust layer probably refer to the application and not to some language (e.g., the application could prove some statement by using deductive reasoning and a statement could be trusted if it is proven and digitally signed by some trusted third party and even encrypted). The user would very likely play an important role in the Trust layer, because it is the user that should decide whether or not some information source should be trusted. These further layers may enhance Ontology languages in the future.

### 3.1.2 Ontology and Web Ontology languages

As shown in the SW architecture (cf. Figure 3.1), URIs, Unicode and XML are considered as the basic blocks on which lie RDF(S) and the higher layer of Ontology.

In AI and Web research, an *ontology* is introduced as "a formal explicit specifica-

[4]Modeling primitives are the basic building blocks for modeling.

[5]*A taxonomy* defines classes of object and the relations among them.

tion of a shared conceptualization" [57]. A *conceptualization* refers to an abstract model of some phenomenon in the world that identifies the relevant concepts of the phenomenon. An ontology is a *specification* because it represents the conceptualization in a concrete form. *Explicit* means that the type of concepts used and the constraints on their use are explicitly defined. *Formal* refers to the fact that the ontology should be machine-understandable. *Shared* indicates that an ontology captures consensual knowledge, that is, it is not restricted to one individual but accepted by a group.

According to this definition, an ontology must include a vocabulary that is used to formalize the knowledge in the ontology, and corresponding definitions that define the semantics of the vocabulary terms. Typically, the vocabulary includes terms for concepts, relations and individuals, while the descriptions of these terms are specified by axioms or assertions:

- *Concepts* (or *classes*) represent sets of objects with common characteristics within the domain of interest;
- *Relations* (or *properties*) represent relationships among concepts;
- *Individuals* are individual objects in the domain of interest; When an individual is a member of a class, it can be called an *instance* of the class.
- *Axioms* are sentences that are always true and are used in general to enforce suitable characteristics of concepts, relations;
- *Assertions* (or *facts*) are sentences that are always true and are used to enforce suitable characteristics of individuals.

The axioms and assertions in an ontology are normally constructed by the *constructors*. Constructors are the building blocks and specify the expressive power of an Ontology language, which is used to represent ontologies. In order to construct more expressive formal definitions, more relevant constraints (expressed by using constructors) among vocabulary terms are needed. For example, classes, subclasses and relations among objects are a very powerful tool for Web use. We can express a large number of relations among objects by assigning properties to classes and allowing subclasses to inherit such properties. Inclusion relations in general (i.e., subclass and subproperty) can also present the reusability of ontologies: large ontologies can be constructed by assembling and refining existing components. Ontology languages can represent more meta-information such as cardinality of relationships, the transitivity of relationships, etc. With Web ontology languages, the meaning of terms or of XML codes used on a Web page can be defined by pointers from the page to an ontology.

The early Web Ontology language is SHOE[6]. This language is based on HTML, which is not considered as a building block of the Semantic Web. In SHOE information must be repeated and this redundancy may cause significant maintenance problems. SHOE's data-model is similar to that of RDFS but it does not include the reification mechanism of RDFS (i.e., an RDF statement is used as an object in a triple).

RDFS can be seen as a very-simple web ontology language, with constructors to define classes, properties, inclusion relations for both classes and properties (i.e., subclass and subproperty), and domain and range constraints on properties. However, in RDFS there are no existence or cardinality constraints (e.g., we cannot say that all instances of person have a mother that is also a person, or that persons have exactly two parents) and no transitive, inverse or symmetrical properties (e.g., we cannot say that isPartOf is a transitive property, that hasPart is the inverse of isPartOf or that these properties are mutually symmetrical). There is no distinction between classes and instances. More expressive constructors like axioms cannot be expressed in RDF(S). Therefore it cannot express closed-world assumptions and several other commonly used non-monotonic constructors. Moreover, some parts of RDF(S) vocabularies are not assigned any formal meaning, and in some cases, notably the reification and container vocabularies, it assigns less meaning than one might expect.

OIL (Ontology Interence Language) [51] lays on top of RDF(S). Actually, OIL is not completely layered, because there is a part of RDF(S), which is not a part of OIL. Reification and meta-classes in RDFS are not allowed in OIL, which means that not all valid RDF(S) is valid OIL. OIL has no explicit import mechanism, inadequate support for ontology evolution, and cannot express the synonym of classes or properties either.

DAML (DARPA Agent Markup Language)[7] attempts to combine the best features of other Semantic Web languages, including RDF, SHOE, and OIL. Like OIL, DAML builds upon RDF. However, DAML is still not more expressive than SHOE.

[6]http://www.cs.umd.edu/projects/plus/SHOE/index.html

[7]http://www.daml.org/language/

DAML+OIL [80] is the result of merging DAML-ONT (an early result of DAML) and OIL. In general, what DAML+OIL adds to RDF Schema is the additional ways to constrain the allowed values of properties, and what properties a class may have. Some restrictions in DAML+OIL cannot be expressed in RDF(S), which means that in some cases decidability is lost (namely when cardinality constraints are applied to properties that are transitive, or that have transitive sub-properties). So decidability in DAML+OIL depends on an informal prohibition of cardinality constraints on *non-simple* properties. A major difference between OIL and DAML+OIL is the support for primitive and more complex data types (come directly from XML Schema) in DAML+OIL, where in OIL only the string data type is supported.

As a result of the work of the W3C Web Ontology Working Group, the Ontology layer has now been instantiated with the Web Ontology Language OWL (cf. Section 3.2). OWL distinguishes from DAML+OIL by some features. A new type of property, owl:SymmetricProperty, has been added, which can directly state that properties are symmetric. Qualified number restrictions are not supported in OWL, mostly to keep things simple. Several bugs and omissions in RDF and RDF Schema have now been fixed by the RDF Core Working Group. This allows OWL to use "official" RDF syntax (in contrast to DAML+OIL). For example, RDF now supports cyclical class and property inclusions (using rdf:subClassOf and rdf:subPropertyOf), so the relevant RDF properties can now be used in OWL; RDFS supports multiple rdfs:domain and rdfs:range properties, with both being treated as equivalent to the intersection of the individual constraints. Besides the DAML+OIL style RDF syntax, the OWL specification also includes an abstract syntax, which provides a higher level and less cumbersome way of writing ontologies. It also has the advantage of allowing a more succinct statement of the semantics. The abstract syntax is defined using an extended BNF (Backus Naur Form) notation [94]. A translation from this syntax to the RDF syntax is available.

It is important that the language used to express ontologies is formal and machine-processable. Usually, ontology languages are based on rigorous logical theories, equipped with reasoning algorithms and services. These logic foundations themselves form the formal semantics of ontology languages and provide an inference model for automated reasoning. Indeed, the definition of what an ontology is (a formal explicit specification) suggests that we need a formal, well-understood language in order to make the ontology machine-understandable. For example, SHOE bases on Datalog [54], which is a minor variant of Horn logic [58]. The initial foundation of RDFS language is semantic networks [122], where each concept is represented by a node in a graph, a relation between two concept is presented by an arc in a graph, which is also called a *link*. However, the semantics of these links are often unclear. Therefore W3C has used the set theory to clarify these semantics (e.g. two kinds of arc *is-a* and *instance-of*) and define the formal semantics of RDFS [71]. However, there is no inference model provided for this language. The only inference mechanism is based on the inclusion relation existing between classes. There is actually a mapping from OIL to the $\mathcal{SHIQ}$, a language in the Description Logic family (cf. Section 3.3). The advantages of OIL are tied to its description logic basis. Description logic constructors allow consistency to be checked, which eases the construction of high-quality ontologies. DAML+OIL is equivalent to a very expressive Description Logic $\mathcal{SHIQ}(\mathbf{D})$, which is $\mathcal{SHIQ}$ language with the addition of data types (cf. Section 3.3) [77]. As the successor of DAML+OIL, OWL is also based on Description Logics.

Capturing semantics from the existing information is the main goal for the semantic web languages. But how to represent clear, explicit, machine-understandable and -processable semantics is not a trivial task. It is a clear go-direction for the design of languages, from SHOE to OWL. We will introduce the latter, the current standardized Web Ontology language, in Section 3.2. Description logics, which is used as the logic foundation for this language, will be introduced in Section 3.3.

## 3.2 Web Ontology Language OWL

The challenge of the Semantic Web is to create a language expressive enough to represent the rules for inference and also to allow export rules in other knowledge systems on the Web. DAML+OIL has been accepted as the de facto standard for ontologies for the Semantic Web until the arrival of a new Web Ontology language, OWL [114].

The OWL language is a revision of DAML+OIL incorporating learnings from the design and application use of it. OWL implements layering on top of RDF(S). It is more powerful at the level of expressiveness and easier to share and exchange knowledge in the Semantic Web than DAML+OIL.

A drawback of DAML+OIL is that it becomes more and more complex to read and so it's more and more difficult to interpret and to know if an ontology can be reused or not. OWL addresses this problem by specifying three language "layers", according to increasing expressiveness, namely OWL Lite, OWL DL and OWL Full.

OWL Lite has been defined with the intention of creating a simple language that will satisfy users primarily needing a classification hierarchy and simple constraint features. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. OWL Lite is actually based on the description logic $\mathcal{SHIF}(\mathbf{D})$. But some constructors in $\mathcal{SHIF}(\mathbf{D})$ (e.g. concept disjunction) are not allowed in the syntax of OWL Lite. Having the minimum expressivity in the OWL family, OWL Lite has the lowest formal complexity. However, it is sufficient to represent *thesaurus*[8] and *taxonomies*[9] or other hierarchies with simple constraints. The tools for implementation of OWL Lite and for migration from the existing thesaurus and taxonomies are also less expensive.

OWL DL includes the complete OWL vocabulary, interpreted under a number of simple constraints. The primary among these constraints is type separation. That is, class identifiers cannot simultaneously be properties or individuals. Similarly, properties cannot be individuals. OWL DL has a maximum expressivity that maintains the computational completeness (i.e., all conclusions are guaranteed to be calculated) and decidability (i.e., all calculations eventually finish in a finite time). This is thanks to the underlying language of OWL DL, namely the description logic $\mathcal{SHOIN}(\mathbf{D})$ [79]. It is this description logic language that gave the name to OWL DL. OWL DL is therefore appropriate to represent ontologies in need of high expressivity while maintaining computability and can be viewed as an expressive DL with Web syntax. Furthermore, OWL DL is provided with different syntaxes. The most prominent is the RDF/XML syntax, which is actually used in the language reference. However, the normative syntax for OWL DL is the abstract syntax, described in [81], which is more succinct and more understandable. OWL Lite and OWL DL pose several restrictions on the use of RDF and redefine the semantics of the RDFS primitives. Thus, OWL Lite and OWL DL are not properly layered on top of RDFS.

The most expressive species of OWL is OWL Full. It lays on top of both RDFS and OWL DL. OWL Full includes the complete OWL vocabulary, interpreted more broadly than in OWL DL, with the freedom provided by RDF. In OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. Another significant difference from OWL DL is that a data type property can be marked as an InverseFunctionalProperty. OWL Full is designed for developers, implementers and users who need the maximum expressivity, the freedom of RDF syntax, but with no guarantee of computability. Currently, there

[8]A thesaurus is a listing of words with similar, related, or opposite meanings.

[9]A taxomony is an explicit hierarchy.

is still no reasoning tools or software that are capable of reasoning to complete all the features of OWL Full.

A legal ontology in OWL Lite is also legal in OWL DL and OWL Full. All documents in OWL (Full, DL, Lite) are valid documents in RDF(S) while only a few documents in RDF(S) are legal documents in OWL Lite or OWL DL. Note that all RDF(S) documents are documents in OWL Full. Compared with RDF(S), OWL can express further such as the relations between classes (e.g. disjunction), the cardinality (e.g. "exactly one"), equality, more types of property (e.g., object properties or annotation property), characteristics of properties (e.g., symmetry, transitivity, inverseFunctional, etc.), and enumerated classes.

Example 3.2.1 shows a simple OWL ontology. Its syntax in RDF/XML is shown in Figure 3.3. A typical OWL ontology begins with a namespace declaration. The tag owl:Ontology is used to state the beginning (or header) of the ontology. The rdfs:label entry provides an optional human readable name for an element of the ontology. The ontology header definition is closed with the tag `</owl:Ontology>`. This prelude is ultimately closed by the tag `</rdf:RDF>`.

**Example 3.2.1 (A simple ontology).** *A simple Dormitory ontology may consist of the following elements:*

- *concepts:* Student, Certificate *and* Trainee*;*
- *properties:* hasTrainingCertificate *and* compatriot*;*
- *a background assumption of the domain: Trainee are Student who have training certificate;*

## 3.3 Description Logics

Logic is well known as a way of using rules to make the inference. Adding logic to the Web is already a task before the Semantic Web community. However, the mixture of mathematical and technical decisions complicate this task. Therefore a logic should be built so that it is powerful enough to describe the complex properties of objects in Web ontologies, but not too expressive to cause the undecidable problem[10].

[10]Undecidable problem: there is no algorithm that finishes in a finite time.

```
<rdf:RDF
    xmlns:owl ="http://www.w3.org/2002/07/owl#"
    xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">

<owl:Ontology rdf:about="">
  <rdfs:comment>An OWL ontology example</rdfs:comment>
  <rdfs:label>Dormitory Ontology</rdfs:label>
</owl:Ontology>

<owl:Class rdf:ID = "Student"/> <rdfs:label>Student</rdfs:label>

<owl:Class rdf:ID ="Certificate"/>

<owl:ObjectProperty rdf:ID="hasTrainingCertificate">
    <rdfs:domain rdf:resource="#Student"/>
    <rdfs:range rdf:resource="#Certificate"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="Trainee">
 <rdfs:subClassOf rdf:resource="#Student"/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#hasTrainingCertificate"/>
   <owl:allValuesFrom rdf:resource="#Certificate"/>
  </owl:Restriction>
 </rdfs:subClassOf>
 </owl:Class>

 <owl:ObjectProperty rdf:ID="compatriot">
  <rdf:type rdf:resource="&owl;TransitiveProperty"/>
  <rdfs:domain rdf:resource="#Student"/>
  <rdfs:range rdf:resource="#Student"/>
</owl:ObjectProperty>

</rdf:RDF>
```

Figure 3.3: An OWL ontology in RDF/XML serialization

Description Logics (DLs) [7], a logic-based language family, can provide a high expressivity while guaranteeing the decidable reasoning. Actually, it was designed as a decidable subset of First-Order Logic to represent and reason about the knowledge of an application domain in a structured and unambiguous way, so as knowledge can be retrieved and reused efficiently. Therefore, DLs have been chosen as the logic foundation for many ontology languages as well as for the current standard Web ontology language OWL. In what follows we will give an overview of this family.

### 3.3.1 Syntax and Semantics

The foundations of DLs are concept and role descriptions (or concepts and roles for short). A *concept* represents a class of objects (or *individuals*) sharing some common characteristics. When an individual is a member of a concept, it is called an *instance* of that concept. Note that these notions are the same as for Ontology (presented in Section 3.1.2). A *role* represents a relationship among objects. We present here only DLs which use the standard notation, i.e., roles are binary relations. In terms of ontology, a role is considered as a property. A DL provides a set of operators, called *constructors*, to build more complex descriptions from others.

**Definition 3.3.1 (Concepts).** *Let $N_C$ be a set of concept names. The set of concepts of a DL language $\mathcal{L}$, the so-called set of $\mathcal{L}$-concepts, is the smallest set such that*

1. *every concept name $A \in N_C$ is a concept,*
2. *if $C$ is a concept, then the descriptions resulting from applying the relevant concept constructors too $C$ are also concepts.*
3. *if $C$ is a concept and $R$ is a role, then the descriptions resulting from applying the relevant concept constructors t $C$ and $R$ are also concepts.*
4. *if $C$ and $D$ are concepts and $R$ is a role, then the descriptions resulting from applying the relevant concept constructors to $C$, $D$ and $R$ are also concepts.*

Two special concept names $\top$ (*top*) and $\bot$ (*bottom*) represent the most general concept (i.e. the universe) and the least general concept (i.e. nothing) respectively. Their formal semantics can be seen in Table 3.1.

**Definition 3.3.2 (Roles).** *Let $\mathbf{R}$ be a set of role names. The set of roles of a DL language $\mathcal{L}$ is the smallest set such that*

1. *every role name $R \in \mathbf{R}$ is a role,*
2. *if $R_1$ and $R_2$ are roles, then the descriptions resulting from applying the relevant role constructors to $R_1$ and/or $R_2$ are also roles.*

Table 3.1: Syntax and Semantics of concept descriptions in $\mathcal{ALC}$ language

| Constructor | Syntax | Semantics |
|---|---|---|
| top | $\top$ | $\Delta^{\mathcal{I}}$ |
| bottom | $\bot$ | $\emptyset$ |
| concept name | $A$ | $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ |
| negation | $\neg C$ | $\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$ |
| conjunction | $C \sqcap D$ | $C^{\mathcal{I}} \cap D^{\mathcal{I}}$ |
| disjunction | $C \sqcup D$ | $C^{\mathcal{I}} \cup D^{\mathcal{I}}$ |
| exists restriction | $\exists R.C$ | $\{x \in \Delta^{\mathcal{I}} \vert \exists y.\langle x,y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ |
| value restriction | $\forall R.C$ | $\{x \in \Delta^{\mathcal{I}} \vert \forall y.\langle x,y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$ |

The most common constructors include boolean operators on concepts (e.g., conjunction and disjunction), and quantification over roles. Further constructors that have been considered important include number restrictions, which allow one to state limits on the number of relations that an individual may have via a certain role, and role constructors such as inverse (e.g. $\mathsf{hasIngredient}^{-}$ is the inverse role of hasIngredient and equivalent to the role $isIngredientOf$). Constructors can also be used to build axioms as presented later in this section.

Description Logics have set-based model-theoretic semantics, which is specified in terms of interpretations. An *interpretation* $\mathcal{I}$ is defined as a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the *domain of interpretation* and is a nonempty set of objects, $\cdot^{\mathcal{I}}$ is an *interpretation function*. This function maps each concept $A \in N_C$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, each role name $R \in \mathbf{R}$ to a set of binary relations over the domain $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ (i.e., $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$). The interpretation function can also be extended to give semantics to concept and role descriptions.

**Example 3.3.1 (An $\mathcal{ALC}$-concept: syntax, semantics and interpretation).** *Suppose we want to build a concept in the DL language $\mathcal{ALC}$ [116] to describes the cakes that have cream cheese as an ingredient, following the syntax and semantics in Table 3.1, we can build a description as follows:*

$$\mathit{Cake} \sqcap \exists\, \mathit{hasIngredient.CreamCheese},$$

*where Cake and CreamCheese are concepts, hasIngredient is a role. Following Definition 3.3.1, $\exists$ hasIngredient.CreamCheese and Cake $\sqcap$ $\exists$ hasIngredient.CreamCheese are also concepts. Suppose we have an interpretation domain*

$\Delta^{\mathcal{I}} = \{\mathsf{NYcheeseCake}, \mathsf{St\text{-}Amour}, \mathsf{Madeleine}, \mathsf{Philadelphia}, \mathsf{Neufchatel}\}$,

*and an interpretation function defined as:*

$\mathsf{Cake}^{\mathcal{I}} = \{\mathsf{NYcheeseCake}, \mathsf{St\text{-}Amour}, \mathsf{Madeleine}\}$

$\mathsf{CreamCheese}^{\mathcal{I}} = \{\mathsf{Philadelphia}\}$

$\mathsf{hasIngredient}^{\mathcal{I}} = \{\langle \mathsf{NYcheeseCake}, \mathsf{Philadelphia}\rangle, \langle \mathsf{St\text{-}Amour}, \mathsf{Neufchatel}\rangle\}$

*According to Table 3.1, we have:*

$(\exists\, \mathsf{hasIngredient}.\mathsf{CreamCheese})^{\mathcal{I}} = \{\mathsf{NYcheeseCake}\}$

$(\mathsf{Cake} \sqcap \exists\, \mathsf{hasIngredient}.\mathsf{CreamCheese})^{\mathcal{I}} = \{\mathsf{NYcheeseCake}\}$

$\diamond$

The function $\cdot^{\mathcal{I}}$ can even be extended further to give semantics to *axioms* (i.e. logical sentences over concepts or roles) and *assertions* (i.e. logical sentences over individuals) in a *DL knowledge base.*

**DL Knowledge Bases**. A typical Description Logic knowledge base (KB) consists of two distinct parts: the intensional knowledge (TBox and RBox), i.e. knowledge in the conceptual level, and extensional knowledge (ABox), i.e. knowledge in the individual level.

Intuitively, an R(ole)Box is a set of role axioms, describing how roles are related to each other. For example, we can use role axioms to describe that the role hasPerfume is a kind of the relationship hasSmell. Sevaral DL languages, e.g. $\mathcal{ALC}$, do not provide role axioms. In this case, RBox is disregarded in a DL knowledge base. However, for modern DL languages (e.g. $\mathcal{SHOIN}$), the role hierarchy is important and so RBox is an indispensable component of a DL knowledge base. An RBox is formally defined as follows.

**Definition 3.3.3.** *An RBox $\mathcal{R}$, a so-called role hierarchy, is a finite, possibly empty, set of role inclusion axioms of the form $R_1 \sqsubseteq R_2$, where $R_1, R_2$ are roles. A role equivalence axiom of the form $R_1 \equiv R_2$ is an abbreviation for $R_1 \sqsubseteq R_2$ and $R_2 \sqsubseteq R_1$.*

*An interpretation $\mathcal{I}$* satisfies *an inclusion axiom $R_1 \sqsubseteq R_2$, or $\mathcal{I}$* models *$R_1 \sqsubseteq R_2$ (written as $\mathcal{I} \models R_1 \sqsubseteq R_2$), if ${R_1}^{\mathcal{I}} \subseteq {R_2}^{\mathcal{I}}$; it* satisfies *a role equivalence $R_1 \equiv R_2$ (written as $\mathcal{I} \models R_1 \equiv R_2$) if ${R_1}^{\mathcal{I}} = {R_2}^{\mathcal{I}}$. An interpretation $\mathcal{I}$* satisfies *an RBox*

*$\mathcal{R}$ iff it satisfies all the axioms in $\mathcal{R}$. Such an interpretation is called a* model *of $\mathcal{R}$ (written as $\mathcal{I} \models \mathcal{R}$).*

A T(erminological)Box is a set of concept axioms, describing how concepts are related to each other. Figure 3.4 is an example of Tbox, where the concept axiom $\mathsf{CheeseCake} \sqsubseteq \mathsf{Cake} \sqcap \exists \mathsf{hasIngredient}.\mathsf{Cheese}$ says that only $\mathsf{Cake}$ that has the ingredient of $\mathsf{Cheese}$ can be considered as $\mathsf{CheeseCake}$. Formally, a TBox is defined as follows.

**Definition 3.3.4.** *A TBox $\mathcal{T}$ is a finite, possibly empty, set of general concept inclusion (GCI) axioms of the form $C \sqsubseteq D$, where $C$ and $D$ are concepts. A concept equivalence axiom of the form $C \equiv D$ is an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$.*

*An interpretation $\mathcal{I}$* satisfies *a GCI $C \sqsubseteq D$, or $\mathcal{I}$* models *$C \sqsubseteq D$ (written as $\mathcal{I} \models C \sqsubseteq D$), if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$; it* satisfies *a concept equivalence $C \equiv D$ (written as $\mathcal{I} \models C \equiv D$), if $C^{\mathcal{I}} = D^{\mathcal{I}}$. An interpretation $\mathcal{I}$* satisfies *a TBox $\mathcal{T}$ iff it satisfies all the axioms in $\mathcal{T}$. Such an interpretation is called a* model *of $\mathcal{T}$ (written as $\mathcal{I} \models \mathcal{T}$). An interpretation $\mathcal{I}$* satisfies *a TBox $\mathcal{T}$ (with respect to (w.r.t) an RBox $\mathcal{R}$) iff it satisfies $\mathcal{T}$ and $\mathcal{R}$. In this case, $\mathcal{I}$ is a* model *of $\mathcal{T}$ w.r.t $\mathcal{R}$ (written as $\mathcal{I} \models_{\mathcal{R}} \mathcal{T}$) and $\mathcal{T}$ is* satisfiable *w.r.t $\mathcal{R}$.*

$$\boxed{\begin{array}{l} \mathsf{CreamCheese} \sqsubseteq \mathsf{Cheese} \\ \mathsf{CheeseCake} \sqsubseteq \mathsf{Cake} \sqcap \exists \mathsf{hasIngredient}.\mathsf{Cheese} \\ \mathsf{AmericanCheeseCake} \sqsubseteq \mathsf{Cake} \sqcap \exists \mathsf{hasIngredient}.\mathsf{CreamCheese} \end{array}}$$

Figure 3.4: A Tbox example

The third component of a DL knowledge base is the A(ssertional)Box. Abox allows to describe a specific state of the world by introducing individuals and assertions over them (also called individual axioms). For example, in the Abox in Figure 3.5, it is asserted that $\mathsf{NYcheeseCake}$ is an instance of $\mathsf{Cake}$, $\mathsf{Philadelphia}$ is an instance of $\mathsf{CreamCheese}$ and that the relationship between $NYCheeseCake$ and $\mathsf{Philadelphia}$ is an instance of the role $\mathsf{hasIngredient}$. Formally, an Abox is defined as follows.

**Definition 3.3.5.** *An ABox $\mathcal{A}$ is a finite, possibly empty, set of assertions of the form $a : C$ and $(a, b) : R$, where $C$ is a concept, $R$ is a role, $a$ and $b$ are individual names.*

*An interpretation $\mathcal{I}$* satisfies *an assertion $a : C$ (written as $\mathcal{I} \models a : C$) if $a^{\mathcal{I}} \in C^{\mathcal{I}}$; it* satisfies *an assertion $(a, b) : R$ (written as $\mathcal{I} \models (a, b) : R$) if $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$. An* *interpretation $\mathcal{I}$* satisfies *an ABox $\mathcal{A}$ iff it satisfies all the assertions in $\mathcal{A}$. Such an interpretation is called a* model *of $\mathcal{A}$ (written as $\mathcal{I} \models \mathcal{A}$). An interpretation $\mathcal{I}$* satisfies *an ABox $\mathcal{A}$ w.r.t a TBox $\mathcal{T}$ and an Rbox $\mathcal{R}$ if it is a model of $\mathcal{A}$ and $\mathcal{T}$ and $\mathcal{R}$. Such an Abox is called* consistent *w.r.t $\mathcal{T}$ and $\mathcal{R}$.*

$\mathsf{NYcheeseCake} : \mathsf{Cake}, \quad \mathsf{St\text{-}Amour} : \mathsf{Cake}, \ \mathsf{Philadelphia} : \mathsf{CreamCheese}$
$(\mathsf{NYcheeseCake}, \mathsf{Philadelphia}) : \mathsf{hasIngredient}$
$(\mathsf{St\text{-}Amour}, \mathsf{Neufchatel}) : \mathsf{hasIngredient}$

Figure 3.5: An Abox example

Individual names can also be used to form concepts called *nominals*, which are interpreted as sets consisting of exactly one individual. The DL language introduced in Chapter 4 provides this feature.

Usually, Abox is assumed to be an *open world*. It implies that one cannot assume that the knowledge in a knowledge base is complete. This is derived from the fact that a knowledge base may have many models, only some features of which are constrained by the assertions. For example, the assertion $(\mathsf{St\text{-}Amour}, \mathsf{Neufchatel}) : \mathsf{hasIngredient}$ expresses that Neufchatel is an ingredient of St-Amour in all models; in some of these models, St-Amour is only made of Neufchatel, while in others, St-Amour may be made of some other ingredients. Another assumption about ABox is also often used, namely the *unique name assumption* (UNA). UNA means that if $a$ and $b$ are two distinct individual names, then $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. In some languages providing the nominal constructor (mostly in presence of number restrictions and inverse role), this assumption is omitted, i.e., two nominals can refer to the same individual (see Chapter 5).

Now we define formally a DL knowledge base.

**Definition 3.3.6.** *A knowledge base $\Sigma$ is a triple $\langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{R}$ is an RBox, $\mathcal{T}$ is a TBox, and $\mathcal{A}$ is an ABox.*

*An interpretation $\mathcal{I}$* satisfies *a knowledge base $\Sigma$ if it satisfies $\mathcal{R}, \mathcal{T}$ and $\mathcal{A}$. Such an interpretation is called a* model *of $\Sigma$ (written as $\mathcal{I} \models \Sigma$) and $\Sigma$ is said to be* satisfiable. *We say that $\Sigma$* logically implies *$\alpha$, where $\alpha$ is either an assertion or an axiom if $\alpha$ is satisfied by every model of $\Sigma$ (written as $\Sigma \models \alpha$).*

DL languages are characterized by the constructors they provide. Each DL language is usually named according to its set of constructors. For example, the language $\mathcal{S}$, often used as a minimal language in the modern DLs, extends $\mathcal{ALC}$ with transitive roles (e.g. *hasAncestor*). $\mathcal{S}$ got its name because it relates to the propositional (multi)modal logic $S4$ [115], and because adding letters to represent additional constructors became cumbersome. The language $\mathcal{SI}$ extends $\mathcal{S}$ with *i*nverse roles; $\mathcal{SHI}$ extends $\mathcal{SI}$ with role *h*ierarchy; $\mathcal{SHIF}$ extends $\mathcal{SHI}$ with *f*unctional number restrictions, etc. Besides, DL languages can also be extended with *d*atatypes, such as $\mathcal{SHOIN}(\mathbf{D})$. In this case, roles are categorized as *abstract* roles (the relationship between two objects) and *concrete* roles (the relationship between an object and a data type).

## 3.3.2 Reasoning Services

A knowledge base may contain implicit knowledge that can be made explicit through reasoning. A DL system not only stores axioms and assertions, but also offers services that reason about them. Typically, reasoning with a DL knowledge base is a process of discovering implicit knowledge entailed by the knowledge base. For example, the assertion $\mathsf{NYcheeseCake} : \mathsf{AmericanCheeseCake}$ can be deduced from the TBox and Abox in Figure 3.4 and 3.5 respectively.

Reasoning services can be roughly categorized as basic services and complex services, which are built upon basic ones. For example, in order to check whether a domain model is correct, or to optimize queries that are formulated as concepts, we may want to know whether some concept is more general than another one. This service is, thus, a basic one and is called the *subsumption* problem. Let $\mathcal{L}$ be a Description Logic, $\Sigma$ be a knowledge base, $\mathcal{T}$ be the Tbox, $\mathcal{R}$ be the Rbox, and $C, D$ be concepts in $\mathcal{L}$, $a$ be an individual name. The principal basic reasoning services include:

*Knowledge Base Satisfiability.* A knowledge base $\Sigma$ is said to be *satisfiable*, written as $\Sigma \not\models \bot$, if there exists an interpretation $\mathcal{I}$ that satisfies $\Sigma$. Knowledge Base Satisfiability is the problem of checking if $\Sigma \not\models \bot$ holds, i.e., whether there exists a model $\mathcal{I}$ of $\Sigma$.

*Concept Satisfiability.* A concept $C$ is *satisfiable* w.r.t. $\mathcal{R}$ and $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a *model* of $C$ w.r.t. $\mathcal{R}$ and $\mathcal{T}$. Concept Satisfiability is the problem of checking if there exists such a model.

*Subsumption.* A concept $D$ *subsumes* a concept $C$ w.r.t. $\mathcal{R}$ and $\mathcal{T}$ (written as $C \sqsubseteq_{\mathcal{R},\mathcal{T}} D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds in every model $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$. Two concepts $C$, $D$ are *equivalent* w.r.t. $\mathcal{R}$ and $\mathcal{T}$ (written as $C \equiv_{\mathcal{R},\mathcal{T}} D$) if they subsume each other w.r.t. $\mathcal{R}$ and $\mathcal{T}$. Subsumption checking is the problem of checking if in every model $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Algorithms that check subsumption are also employed to calculate the concept hierarchy (i.e. taxonomy) of a knowledge base.

*Instance Checking.* This service is to check if an individual is an instance of a specific concept. An individual name $a$ is an instance of a concept $C$ w.r.t $\Sigma$ iff in every model $\mathcal{I}$ of $\Sigma$ we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$.

The basic reasoning services mentioned above are not independent of one another. With the presence of the negation constructor in DL languages, all of these services can be reduced to knowledge base satisfiability [105]. Hence, KB satisfiability can be regarded as the most general of the mentioned reasoning problems. As we will see later in this section, for some DLs, it is also possible to polynomially reduce KB satisfiability to concept satisfiability.

Concept satisfiability and subsumption are reasoning problems that are usually considered only w.r.t. an Rbox and a TBox rather than a knowledge base. The reason is that the ABox does not interfere with these problems as long as the KB is satisfiable. In case there is the presence of nominals, Aboxes can be captured using nominals. Therefore, it suffices to consider satisfiability of TBoxes and Rboxes. More precisely, we have the following theorems (presented slightly differently from the original versions):

**Theorem 3.3.1.** *[112] If $\mathcal{L}$ is a DL language that does not provide the nominal constructor, and $\Sigma = \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$ is a satisfiable knowledge base in $\mathcal{L}$, then for every pair of $\mathcal{L}$-concepts $C$, $D$ we have $\Sigma \models C \sqsubseteq D$ iff $C \sqsubseteq_{\mathcal{R},\mathcal{T}} D$.*

**Theorem 3.3.2.** *[130] If $\mathcal{L}$ is a DL language that provides the nominal constructor, and $\Sigma = \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle$ is a satisfiable knowledge base in $\mathcal{L}$, then knowledge base satisfiability can be polynomially reduced to satisfiability of $\mathcal{R}$ and $\mathcal{T}$.*

Moreover, the concept satisfiability and subsumption problems can be reduced to each other. The reason is that $C$ is unsatisfiable w.r.t an Rbox $\mathcal{R}$ and a Tbox $\mathcal{T}$ iff $C \sqsubseteq_{\mathcal{R},\mathcal{T}} \bot$. Additionally, in the presence of the negation constructor in DL languages , $C \sqsubseteq_{\mathcal{R},\mathcal{T}} D$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t an Rbox $\mathcal{R}$ and a Tbox $\mathcal{T}$. Therefore, the undecidability of one of these services implies that none of the above basic reasoning services are computable.

A concept axiom $C \sqsubseteq D$ is *definitional* if $C$ is a concept name. In this case, the axiom is called the *definition* of $C$. Tbox is called *simple* if it is *acyclic* (i.e., there are no cyclic dependencies between the axioms in the Tbox), and contains only *unique* definitional axioms (i.e., $\mathcal{T}$ contains only one definition for each concept name). In this case, concept satisfiability w.r.t. $\mathcal{R}$ and $\mathcal{T}$ can be reduced to concept satisfiability w.r.t. $\mathcal{R}$ by a process called *unfolding.* The *unfolding* $C_{\mathcal{T}}$

of concept $C$ w.r.t. $\mathcal{R}$ and a simple Tbox $\mathcal{T}$ is obtained by successively replacing every defined name in $C$ by its definition in $\mathcal{T}$ until only primitive (i.e., undefined) names occur. Since $C_\mathcal{T}$ is obtained from $C$ in such a way that both are interpreted in the same way in any model of $\mathcal{T}$ and $\mathcal{R}$, it follows that $C_\mathcal{T} \equiv_{\mathcal{R},\mathcal{T}} C$. Hence, $C$ is satisfiable w.r.t. $\mathcal{R}$ and $\mathcal{T}$ iff $C_\mathcal{T}$ is satisfiable w.r.t. $\mathcal{R}$ and $\mathcal{T}$. Since $C_\mathcal{T}$ contains no defined names, $C_\mathcal{T}$ is satisfiable w.r.t. $\mathcal{R}$ and $\mathcal{T}$ iff it is satisfiable w.r.t. $\mathcal{R}$. As a result, $C$ is satisfiable w.r.t. $\mathcal{R}$ and $\mathcal{T}$ iff it is satisfiable w.r.t. $\mathcal{R}$. However, to reduce the size of the search space, a favorable technique is *lazy unfolding* that performs the unfolding of $C$ w.r.t. $\mathcal{R}$ and $\mathcal{T}$ on demand [96].

When a Tbox $\mathcal{T}$ includes arbitrary CGIs (i.e., a *general* TBox), the reasoning w.r.t. $\mathcal{T}$ and $\mathcal{R}$ can also be reduced to reasoning only w.r.t $\mathcal{R}$ by using a technique called *internalization* [5]. Internalization works by testing the satisfiability of a new concept which incorporates (i.e. internalizes) all CGIs in $\mathcal{T}$ and the concept the satisfiability of which needs to be verified. More precisely, reasoning in the DL languages providing transitive and hierarchy role uses the internalization specified in the following theorem (slightly different from the original version).

**Theorem 3.3.3.** *[87] Let $\mathcal{L}$ be a DL language providing general negation, $\mathcal{R}$ be an Rbox providing transitive roles, $\mathcal{T}$ be a Tbox in $\mathcal{L}$ and $C$, $D$ be $\mathcal{L}$-concepts, and let*

$$C_\mathcal{T} = \bigsqcap_{C_i \sqsubseteq D_i \in \mathcal{T}} \neg C_i \sqcup D_i,$$

*$\Re$ be a transitive role with $R \sqsubseteq \Re$ $\forall R$ that occurs in $\mathcal{T}$, $C$, $D$, or $\mathcal{R}$. $\Re$ is called the* universal role. *We set*

$$\mathcal{R}_\Re := \mathcal{R} \cup \{R \sqsubseteq \Re | R \textit{ occurs in } \mathcal{T}, C, D, \textit{ or } \mathcal{R}\}.$$

*$C$ is satisfiable w.r.t $\mathcal{R}$ and $\mathcal{T}$ iff $C \sqcap C_\mathcal{T} \sqcap \forall\Re.C_\mathcal{T}$ is satisfiable w.r.t $\mathcal{R}_\Re$. $C \sqsubseteq_{\mathcal{R},\mathcal{T}} D$ iff $C \sqcap \neg D \sqcap C_\mathcal{T} \sqcap \forall\Re.C_\mathcal{T}$ is unsatisfiable.*

Checking satisfiability of concepts is a key inference. As we have seen, a number of other important inferences can be reduced to the concept (un)satisfiability.

## 3.3.3 Reasoning Algorithms

As mentioned in Section 3.3.2, even though knowledge base satisfiability is the most general standard reasoning problem, it is worthwhile to consider solutions for the less general problems. In this section, we will briefly present how to provide inferences for concept satisfiability and subsumption.

Since most DLs[11] are within the two variable fragment of first-order predicate logic, one can think of reducing these two reasoning problems to the known inference problems of a first-order logic language, e.g., $\mathcal{L}^2$ or $\mathcal{C}^2$. However, the complexity of the decision procedures obtained this way is usually higher than necessary [9]. This approach, therefore, should be used to obtain upper bound complexity results of the reasoning problems, rather than to provide reasoning services.

In the early days, subsumption problem was usually addressed by so-called *structural subsumption algorithms*, i.e., algorithms that compare the syntactic structure of concepts. While usually very efficient, these algorithms are only complete for rather simple languages with little expressivity. In particular, DLs with (full) negation and disjunction cannot be handled by this approach. For such languages, tableau-based algorithms (or the so-called *tableau algorithms*) have turned out to be very useful. Instead of directly testing subsumption of concepts, this approach employs negation to reduce subsumption to concept (un)satisfiability. The idea of tableau algorithms is using a tree to represent the model being constructed. The first tableau algorithm for DLs was presented by Schmidt-Schau and Smolka [116] for satisfiability of $\mathcal{ALC}$-concepts. Since then, this approach has been employed to obtain sound and complete satisfiability (and thus also subsumption) algorithms for a great variety of DLs extending $\mathcal{ALC}$ (e.g. [73, 6, 79, 83, 84]).

To be useful for applications, the inferences should be sound, so that every drawn conclusion is correct. It is also desirable to have complete inference, so that every correct conclusion can be drawn. Obviously, these requirements may be in conflict, because a greater expressivity of a DL makes sound and complete inference more difficult or even undecidable. Initially, the emphasis was on the reasoning services of tractable DLs, with an upper bound of polynomial complexity [24]. Unfortunately, only very primitive DLs are "tractable" in this sense, e.g., the satisfiability of $\mathcal{ALC}$-concepts w.r.t. general TBox is already ExpTime-complete [130]. Interestingly, although the theoretical complexity results are discouraging, experimental analysis have shown that worse-case intractability rarely occurs in practice [72, 111, 78]; even some simple optimisation techniques could lead to significant improvement in the experimental performance of a DL system [8]. More recent systems, such as FaCT [75], Pellet [38] and RACER [60], have demonstrated that even with expressive DLs (e.g. $\mathcal{SHOIQ}$), highly optimised implementations

[11]A few exceptions such as the DL introduced by Calvanese et al. [27, 31]

can provide acceptable performance in realistic applications, moving boundaries of "tractability" to somewhere very close to ExpTime-hard, or worse [45].

## 3.4 Integrating Relational Data Sources into the Semantic Web

Although integrating semantics from relational data sources has been interested since a long time ago (e.g. data warehousing, global information systems) [44], data integration from these sources into the Semantic Web is still a new trend of research. This integration is meaningful only when the semantics of relational data sources can be accessed from the Semantic Web environment. Reviewing the works (both in theory and in application development) to address this problem, we can classify them into two categories: query-based and semantic annotation approaches.

### 3.4.1 Query-based Approach

Many of the early as well as the recent works whose goal (or perspective) is to integrate relational data into the Semantic Web employ queries to retrieve semantics from sources [32, 126, 62, 36, 20]. In general, queries created by some way are sent directly to relational databases to retrieve the information corresponding to some notion in a data integration system. The correspondence is built by mapping rules between the relational database schemas of sources and the common data model of an integration system. Typically, these rules are predefined by mapping definers, who are usually human domain experts.

*A theory foundation* for query-based approach was introduced by Calvanese and his colleagues [32]. They proposed a general and formal framework **OIS** (Ontology Integration System) that employs the global schema (cf. Chapter 2). OIS assumed that the source ontology is an arbitrary logical theory and therefore was also applied to data integration [95]. The framework does not make explicit any of the mechanisms proposed, but employs the notion of GAV and LAV to retrieve the semantics from sources. GAV (Global-As-View) associates each term in the global schema to one query over the source relations, while LAV (Local-As-View) reformulates a query on a global schema in terms of queries on the sources. LAV requires a reasoning step in query reformulation process to infer how to use the sources for answering the query. In a case study of LAV, the authors employed $\mathcal{DLR}$, a description logic supporting n-ary roles, for the global ontology language.

In another LAV example [30], the global ontology is expressed in DL-Lite, a restricted DL formalism of $\mathcal{ALCIQ}$ stemmed from representation formalisms developed in ER, UML and Web ontology languages. Therefore DL reasoners can be profited to answer queries.

While GAV approach is considered to allow for a simple query processing strategy, LAV is preferred for the reason that this approach better supports a dynamic environment and thus the Semantic Web.

Recently, LAV is applied in a practical paradigm by Chen and his colleagues [36]. They introduced the **Dartgrid**, a mediated schema-based integration framework to facilitate the integration of heterogeneous relational databases using semantic web technologies. The shared (or domain) ontology is represented in RDF(S). The semantics of relational data sources are mapped manually to the domain ontology by database providers. Mapping rules are described with Datalog-like syntax [54]. Each relational table is mapped to a view over the domain ontology. A typical view consists of two parts: the view head is a relational predicate; the view body is a set of RDF triples.

Halevy et al. [62] introduced **Piazza** peer data management system, which can be seen as a P2P-based data integration framework with consideration of the semantic web vision. Each peer defines its own relational peer schema whose relations are called "peer relations". A query will be posed over the relations from a specific peer schema. A peer can have the data sources or not (in case of the peer playing the role of a mediator). The data sources are defined as "stored relations". In case source schemas are relational models, stored relations are "relations" defined in relational models (cf. Chapter 2). The mapping rules, which are called "storage descriptions", specify the data stored at a peer by mapping each stored relation in the data source as a view over the peer relations. Arguing that most relational, semi-structured, and even RDF data is available in XML form, Halevy et al. have implemented Piazza system with a common XML data model. They defined a mapping language that borrows elements of XML Query. Halevy et al. did not state whether they employed GAV or LAV or not. In their system, mappings are defined as one or more mapping definitions, and they are directional from a source to a target: "we take a fragment of the target schema and annotate it with XML query expressions that define what source data should be mapped into that fragment".

**Tom Barrett et al.** [126] followed also the query-based approach but did not specify LAV/GAV in their method of semantic retrieval. They introduced a solution using also RDF(S) for the representation of all metadata characterizing the content of the information source. Their mediated schema-based integration architecture consists of two major components: Domain Brokers and Resource Brokers. The role of the Domain Broker is query handling. It receives queries from users, distributes queries to the appropriate Resource Brokers, merges the results from various Resource Brokers and passes the combined response back to the user. Every resource has an associated Resource Broker. The role of the Resource Broker is to handle the communication to and from the resource. In particular, it translates queries to the target format (i.e. SQL with relational data sources), converts ontological terms to terms used by the resource (i.e. relational database schemas), and translates responses back into the ontological terms used in the query. Actually, these interchanges are specified by metadata structures defined in RDF(S). That is, mapping rules from resource terms to ontological terms and queries over the domain ontology are described in their RDF-based languages. Based on the mapping rules, the queries over the domain ontology can be translated into SQL queries. The data returned from the source is then translated back to the ontological terms by using mapping rules.

Usually, mapping rules are described by a mapping language. For example, as mentioned above, Piazza employed XML query to define mappings. Tom Barrett et al. introduced an RDF(S)-based language to describe mapping rules. They stated that in their mapping language, mappings are not required to be one-to-one; some property values can be derived; class inheritance is supported and one-to-many relationships between database tables can be modeled in both directions (e.g. part_of and has_part). In Dartgrid, the authors proposed datalog-like syntax built on RDF triples to describe mapping rules. **D2RQ** [20] was presented as a declarative language to describe mappings between relational database schemas and OWL/RDFS ontologies. It is an XML-based language built on experiences gained with D2R MAP [19]. D2RQ wraps one or more local relational databases into a virtual, read-only RDF graph. D2RQ rewrites RDQL[12] queries into SQL queries. The resulting record sets of these SQL queries are transformed into RDF triples that are passed up to the higher layers of the Jena framework [1]. The central object of D2RQ is ClassMap, which is a class or a group of similar classes from ontologies. ClassMap specifies whether instances are identified by URI column value from a database or URI pattern and key value, or blank node. One ClassMap has a set of property-Brigdes specifying how instance properties are created. PropertiyBriges are either DatatypePropertyBridge or ObjectPropertyBridge. Classes and relationships between classes in an ontology can be mapped to tables in a relational model. Each property is mapped to an attribute of the table whose ClassMap has the corre-

[12]RDQL: The query language in Jena system [34], where D2RQ is implemented.

spondent propertyBridge of the property. Actually, D2RQ is less expressive than the language proposed in Dartgrid.

Throughout the works presented above, we can summarize some drawbacks of query-based approach as follows:

1. With regard to GAV approach [32], it is possible that the data retrieved do not obey all semantic conditions in the global ontology, or even no data is retrieved. In this case, the system inconsistency occurs. In the usual case of autonomous, heterogeneous sources, it is very unlikely that the data fit in the global ontology. Therefore, GAV is too restrictive in the sense that the integration system would be often inconsistent. This problem can only be solved if the mapping definers understand well the semantics of the sources or there must be a way to represent declaratively the source semantics (e.g. the semantic annotation approach).

2. Information is exploited directly from relational sources through queries and a set of mappings. Therefore, the expressivity of the mapping language plays the decisive role in the integration capability. Among the mapping languages presented above, Datalog-like language of Dartgrid system is the most expressive. However, it is built on RDF(S) that does not have the underlying inference model. Even creating its own algorithm with consideration of the features of web ontology languages, Dartgrid is enriched not much from inference capabilities of SW technologies for high expressivity (as presented in Section 3.3). Some works have no theoretical foundation for the mapping (such as GAV/LAV) like Barrett et al.

3. Mapping rules are usually predefined by human experts. The manual creation of semantic mapping has long been known to be extremely laborious and error-prone. This can be caused by many reasons as follows [44]:

   - The semantics of the involved elements can be inferred from only a few information sources, typically the creators of data, documentation, and associated schema and data. Extracting semantics information from data creators and documentation is often extremely cumbersome. Frequently, the data creators have long moved, retired, or forgotten about the data. Documentation tends to be sketchy, incorrect, and outdated.

   - Directly manipulating on databases, mapping can only use some clues such as element names, types, data values, schema structures. However, these clues are often unreliable. For example, the name "area" can refer to different real-world entities such as a location or square-feet area of the house. The reverse problem also often holds: two elements with different names (e.g., area and location) can refer to the same real-world entity (the location of the house).

- Moreover, these clues are often incomplete. For example, the name "contact-agent" only suggests that the element is related to the agent. It does not provide sufficient information to determine the exact nature of the relationship (e.g., whether the element is about the agent's phone number or her name).
- To decide the mapping, one must typically examine all elements of the two models to make sure that there is no other mapping better than that. This global nature of mapping adds substantial cost to the mapping process.

Some argument for query-based approach should be that the risk of a possible misunderstanding can usually be minimized through a face to face communication or previous agreement between the exchange partners. Nevertheless, this cannot be accomplished within a volatile environment like the semantic web, where peers may appear and disappear at any time.

4. Moreover, it is possible that the inconsistency of mapping repository occurs (e.g. two mapping rules are in conflict). Inference engine for mapping languages is therefore necessary. However, in all of the works presented above, no mapping language has a logic foundation to support reasoning. Checking the consistency of mapping repository is still an open question.

5. Mapping rules are usually predefined by human experts. Therefore, the semantics captured from data sources are intentional, depending on the application. One application may decide that "email address" corresponds to "contact", another application may decide that it does not. Since query-based approach does not capture the native semantics of data sources, it is only suitable for developing a specific-domain application, but not for exploiting semantics from relational sources on the Semantics Web, where the domain of applications/services is unpredictable (although LAV approach is considered suitable for dynamic environment).

Consequently, it is vital to have a representation format which is unambiguous and available on the Semantic Web to describe the semantics of data sources. That is knowledge representation, the only way to guarantee the faultless interpretation of data on a remote node, because knowledge representation process prevents different sites or partners to interpret the same data differently. The semantic annotation approach follows this idea.

### 3.4.2 Semantic Annotation Approach

In general, the conceptual data models of relational data sources are transformed into ontologies by some mapping rules. The ontologies are represented in some semantic annotation, the so-called ontology language, that is expected to capture all the semantics expressed in data models.

Berners-Lee can be seen as one of the early forerunner in semantic annotation approach [16]. He tried to make a correspondence between relational database schema and RDF: a record is an RDF node; the field (column) name is an RDF property; and the record field (table cell) is a value. His mapping, however, is rather rudimental (e.g. no integrity constraints).

Recently, a few works have proposed semantic annotations for relational data sources on the Web [39, 98, 124, 42, 26, 46].

**RDF Gateway** [39], a platform for data integration, supports not only RDF(S) but also OWL as the common data model. It connects external relational data sources to the Semantic Web via its SQL Data Service. The SQL Data Service queries relational databases to get the schemas required, then maps (translates) the resulting relation schema into an RDFS/OWL ontology. The ontology language (RDFS or OWL) is specified by a parameter called schema_type. For example, a table is mapped to an rdfs:Class if schema_type=rdfs, or to an owl:Class if schema_type=owl. The name of the class is based on the name of the table. Similarly, a column is mapped to a property in RDFS and to a data type property in OWL, using the column name prefixed with the table name. A foreign key is mapped to a property in RDF(S) and to object property in OWL. The domain of the property is the table containing the column corresponding to the property. The range of the property is the class of the referenced table. These rules can be applied automatically to translate relational schemas into the common data model of an integration system. However, these are only some simple rules (one-to-one mapping).

**MAFRA** [119, 98], a peer-to-peer integration system based on a mapping framework for distributed ontologies on the Semantic Web, brings relational databases to the "structural level" of simple ontologies in RDF(S). This is realized semi-automatically by their LiFT tool [138]. LiFT focuses on minimizing syntactical and structural representation heterogeneity. It tries to capture the semantics of relational data sources from an extended-relational model including the traditional relational model and the foreign key constraint. This model is retrieved from DDL definitions[13]. Generally, LiFT translates relations into concepts; attributes into data type properties (i.e. XML Schema data types). The domain of attributes is the concept, which has been created for the relation upon which the attribute is defined. Foreign keys are translated into properties (i.e. object properties in OWL terminology). The translation is performed in two phases. In the first phase concepts are established. In the second phase properties of concepts are created from the database. The mapping rules are checked in sequential order on each relation.

Even the translation tried to remove redundant information in DDL definitions and in their relational models, many drawbacks of LiFT were presented by the authors. The mapping rule whose preconditions are applied is first chosen for the respective relation. This is due to the fact that their mapping rules might be ambiguous. Eventually, the translation process should be influenced and directed by the user to choose the appropriate rule for individual relations. This solution, however, is still not implemented in their current version. In addition, limited by the RDF(S) language, the translation for cardinalities and inverse relations could not be implemented. The semantic intention behind a given relational structure cannot be fully captured either. For example, the translation cannot decide which relation represents the sub entity and which one represents the super entity. Besides, we see that primary keys are not captured in LiFT and there is no reasoning services. The latter is due to the fact that there is no inference model for RDF(S) (cf. Section 3.1.1).

**N. Stojanovic and R.Volz** [124] proposed also a solution for representing the semantics of relational databases in RDF(S). They defined a relational model similar to LiFT, retrieved the model from SQL-DDL, then mapped it to frame logic [93], which can then be represented in RDF(S). Thanks to the expressivity of Flogic, more constraints can be translated (e.g. the primary key constraint). The mapping process consists of five steps: capture information from physical models; analyze the obtained information; form the ontology by applying mapping rules and removing redundant information; evaluate, validate and refine the obtained ontology; and form a knowledge base. All these steps are realized in semi-automatic way. However, encountering the same problem as LiFT, the authors argued that the process could not be completely automated because some ambiguous situations can

[13]The authors employed DatabaseMetaData interface offered by the Java JDBC standard to avoid different SQL DDL dialects imposed by individual databases

arise where applying several rules. Therefore, user interaction is necessary when this kind of ambiguities occurs and domain semantics cannot be inferred. Besides, frame logic is in general undecidable. The authors did not specify the fragment of frame logic they used and how to convert their representation in frame logic into RDF syntax.

**The project of Buccella et al.** [26] proposed a hybrid architecture and a method, based on the use of OWL ontologies, to integrate relational data sources. Each data source has a source ontology that is built in two steps. The first step is (semi)automatically generating OWL initial ontology from physical data models represented in SQL/DDL. SQL/DDL code is analyzed in the order the tables have been created. Tables without foreign keys are explored first. Each CREATE TABLE sentence is analyzed to find the table name and attributes, which are then converted to classes and data type properties respectively in OWL. One-to-many, many-to-many and weak-entity relationships are transformed into OWL classes, properties and restrictions. Actually, tables are translated into classes; attributes are translated into data type properties if they are not the foreign keys, otherwise into object properties. The domain of a data type property is not specified in order to employ the property for many classes while the object properties have the predefined domain and range. Simple CHECK constraints and CREATE DOMAIN are translated to new classes describing the constraints. NOT NULL constraint is translated to the number restriction. In case SQL/DDL code does not show the minimal cardinality, human experts should be needed to add this cardinality after the ontology is built. The second step to build the source ontology is adding semantics, which allows human experts to add restrictions, classes and/or properties to the initial ontology. The authors argued that by knowing the domain of data sources and understanding the structures, the expert could provide more semantics to the ontology.

Although the high expressive ontology language OWL is used to represent the semantics of relational sources, their translation is not capable of expressing the primary keys, even the simple ones[14]. Let us see one of their examples: the table MUSEUM($\underline{\text{NAME}}$, ADDRESS) (the primary key is NAME, ADDRESS not NUL) is translated into DL formalism[15] as follows :

[14]A simple key consists of only one attribute

[15]We rewrite their original version in OWL language in DL formalism for concise and more easily understanding purpose. The symbol "=" is the abbreviation of "$\leq$" and "$\geq$".

$$\begin{array}{ll}
\top & \sqsubseteq \leq 1\ \mathsf{ADDRESS} \\
\top & \sqsubseteq \leq 1\ \mathsf{NAME} \\
\mathsf{MUSEUM} & \sqsubseteq \forall\ \mathsf{ADDRESS.String} \\
\mathsf{MUSEUM} & \sqsubseteq = 1\ \mathsf{ADDRESS} \\
\mathsf{MUSEUM} & \sqsubseteq \forall\ \mathsf{NAME.String} \\
\mathsf{MUSEUM} & \sqsubseteq = 1\ \mathsf{NAME}
\end{array}$$

With this description, it is possible that two MUSEUMs have the same NAME. Consequently, the meaning of primary key constraint is missing. Even though the system allows users to add semantics to the resulting ontology, the original semantics of the sources is already lost and human experts cannot always recover it. Essentially, the approach of Buccella et al. encountered the same problem for query-based approach. That is, semantics added by human experts may not be correct because of human-errors, which can be caused by many reasons explained above. This is due to the fact that they used SQL/DDL as the data models to exploit the semantics of relational sources. Their semi-automatic solution with two steps has shown the disadvantage of SQL/DDL approach, as we have presented in Chapter 2.

Buccella et al. have chosen OWL for many of its advantages, one of which is the reasoning capability. Nevertheless, they did not specify the OWL language used to represent relational sources. If OWL Full is the candidate, their approach cannot support the decidable reasoning.

**Relational.OWL**. In order to share relational data within a Peer-to-Peer environment, C.P. Laborda and Stefan Conrad [42] proposed a language based on OWL, the so-called Relational.OWL. Relational.OWL represents both relational database data and schemas. In particular, tables and columns, primary and foreign keys (including also compound keys[16]), data types and some constraints on columns (i.e. length, scale) are considered in their language. Table 3.2 and 3.3 present the vocabulary of Relational.OWL.

Table 3.2: Classes in Relational.OWL language

| **rdf:ID** | **rdfs:subClassOf** | **rdfs:comment** |
|---|---|---|
| dbs:Database | rdf:Bag | The class of databases |
| dbs:Table | rdf:Seq | The class of database tables |
| dbs:Column | rdfs:Resource | The class of database columns |
| dbs:PrimaryKey | rdf:Bag | The primary key of a table |

[16]Compound keys are the keys consisting of more than one attributes

Table 3.3: Properties in Relational.OWL language

| **rdf:ID** | **rdfs:domain** | **rdfs:range** | **rdfs:comment** |
|---|---|---|---|
| dbs:has | owl:Thing | owl:Thing | A thing has other things inside |
| dbs:hasTable | dbs:Database | dbs:Table | Set of tables of a database |
| dbs:hasColumn | dbs:Table | dbs:Column | Set of columns of a table |
| dbs:isIdentifiedBy | dbs:Table | dbs:PrimaryKey | Primary key of a table |
| dbs:references | dbs:Columns | dbs:Column | Foreign key relationship |
| dbs:length | dbs:Column | xsd:nonNegativeInteger | Length of an entry in column |
| dbs:scale | dbs:Column | xsd:nonNegativeInteger | Scale of an entry in column |

A relational database schema will be represented with the vocabulary of OWL and of Relational.OWL. For example, the table ADDRESS(<u>ADDRESSID</u>, STREET, ZIP, CITY, COUNTRYID) can be represented as follows:

```
<...>
 <owl:Class rdf:ID="ADDRESS">
  <rdf:type rdf:resource="&dbs:Table"/>
  <dbs:hasColumn rdf:resource="#ADDRESS.ADDRESSID"/>
  <dbs:hasColumn rdf:resource="#ADDRESS.STREET"/>
  <dbs:hasColumn rdf:resource="#ADDRESS.ZIP"/>
  <dbs:hasColumn rdf:resource="#ADDRESS.CITY"/>
  <dbs:hasColumn rdf:resource="#ADDRESS.COUNTRYID"/>
  <dbs:isIdentifiedBy>
   <dbs:PrimaryKey> <dbs:hasColumn rdf:resource="#ADDRESS.ADDRESSID"/>
   </dbs:PrimaryKey>
  </dbs:isIdentifiedBy>
 </owl:Class>

 <owl:DatatypeProperty rdf:ID="ADDRESS.COUNTRYID">
  <rdf:type rdf:resource="&dbs;Column"/>
  <rdfs:domain rdf:resource="#ADDRESS"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
  <dbs:references rdf:resource="#COUNTRY.COUNTRYID"/>
 </owl:DatatypeProperty>
<...>
```

Actually, what the authors proposed is not to capture the semantics described in relational models but to rewrite them in another annotation, which is based on RDF(S). In other words, they did not describe the real-world semantics in databases. The semantics, however, could be captured if their annotation, which incorporates Relational.OWL and OWL, has the formal semantics. Nevertheless, the resulting schema representation belongs to OWL Full. There is neither formal semantic foundation underlying Relational.OWL, nor decidable inference model proposed for this kind of annotation.

**OntoGrate** [46] is an ontology-based data integration framework that can be applied to both peer-to-peer and schema-mediated paradigms. In order to represent database schemas in the Semantic Web, the authors extended the expressivity of Web ontology languages. In particular, they introduced an expressive ontology language Web-PDDL, an extension of PDDL (Planning Domain Definition Language) [100] that is based on the first order logic language and has Lisp-like syntax. Web-PDDL [101] is built on top of RDF and employs mapping rules to describe the structure and semantics of data sources in ontologies. A few simple rules were proposed to translate relational models and integrity constraints. In particular, relations are transformed into types; attributes are transformed into predicates; integrity constraints (e.g. foreign key relationships) are transformed into axioms (i.e. rules); and primary keys are transformed into facts.

In order to support the latest standard Web ontology language OWL, the authors built an automatic translator between Web-PDDL and OWL (called PDDOWL translator). In this case, Web-PDDL is the internal language used for ontology integration. Users will interact with the system via OWL language. Types in Web-PDDL will be translated into OWL classes ; predicates will be translated into properties (data type properties for non-foreign key attributes). Axioms (representing integrity constraints that includes foreign key relationships) were said to be translated into RDF syntax [101]. However, facts (representing primary keys) are not considered.

Ontograte exposed some shortcomings. While the transformation was designed to work automatically, the author believed that further work was required to capture more subtle database semantics. User interaction may be required to capture subtle semantics from databases. As analyzed in Chapter 2, relational models are not a good choice for extracting semantics. Additionally, integrity constraints are not transformed into OWL ontology but into RDF. Besides, the authors have defined a particular Web ontology language that is not familiar in the Web semantic community and is not considered as a standard Web ontology language. As the authors argued, some expressions in Web-PDDL cannot be represented in DAML+OIL [101]. And OWL, in its turn, is a restricted version of DAML+OIL (cf. Section 3.2). Consequently, even one more layer is added to translate Web-PDDL to OWL to facilitate worldwide use; the semantics expressed in Web-PDDL cannot not be fully captured in OWL. A proof is that the primary key constraint is missing in the translation.

Moreover, Web-PDDL itself is undecidable because its semantics is based on First Order Logic (FOL), which is well known to be undecidable. For their use of general-purpose theorem proving, they proposed a way to control the inference complexity through a set of “requirement declarations” that requires domain definers to specify exactly the subset of PDDL the domain requires. This job requires that domain definers (who are domain experts rather than technical experts) must be PDDL experts, which may be impossible. In addition, general-purpose theorem proving is unlikely to be a useful technique on the web, and therefore other special-purpose techniques (such as Horn-clause theorem proving, or inference using description logics) play a key role. Many shortcomings of this approach were also shown in one of their papers [101].

## 3.5 Conclusion and Proposition

The Semantic Web technology, and especially the ontology technology, is the first step to enable data integration on a semantic level both within and across organizational boundaries. Ontologies facilitate knowledge sharing and reuse through their formal (machine-understandable) and real-world (human-understandable) semantics. This is due to the fact that ontologies are built on the logic foundation whose most important technology is Description logics. Thanks to the latter, an ontology not only specifies the domain concepts and their relationships, but also includes the manner in which applications or services are permitted to make use of these concepts. Therefore, ontologies play a key role in making databases interoperate on the Semantic Web.

By using Semantic Web ontologies, data integration systems can benefit from the advantages of the following two features: (i) data is published using common vocabulary and grammar; and (ii) the semantic description of data is preserved in ontologies and ready for inference.

To make database content available in the Semantic Web environment, its meaning represented by the database schema must be expressed in terms of ontology. To this end, recent research has followed two major approaches: query-based techniques and semantic annotations. The former accesses directly to databases and retrieves the semantics of data by employing queries and mapping rules predefined by users. The latter employs an appropriate ontology language in order to precisely describe the data sources and their implicit data constraints through the use of a data model.

Query-based approach exploits information directly from sources. It seems that this is a simpler and more effective way w.r.t the semantic annotation approach. However, query-based approach exposes many drawbacks. Since the semantics of data themselves are not contained inside the relational database schema (cf. Chapter 2), specifying mapping requires usually the interference of humans. This process is often difficult, time-consuming and error-prone. Therefore, mapping repository needs a consistency checking, which has not been investigated yet. Furthermore, it seems difficult to reuse the knowledge retrieved from databases because the semantics captured are subjective, depending on specific application. Semantic annotation approach, on the other hand, allows one to choose a data model appropriate to the semantic extraction and also an appropriate ontology language to represent the semantics staying in such a data model. The semantics, therefore, are original, independent from and reusable by different kinds of application. The major advantages of this approach over the query-based on can be shortly listed as follows:

- Independent semantics
- Exact meaning
- Mapping repository consistency
- Reusability
- Automatic ability
- Support for various services/applications

To capture precisely the original semantics of data sources, the expressivity of the data model chosen is an important factor. As analyzed in Chapter 2, conceptual data models are more suitable for extracting the semantics of data sources from databases than logical and physical models. However, no works shown above employed conceptual data models. Some of them employed a combination of logical and a part of physical models (e.g. MAFRA). Some others employed relational models and some integrity constraints but did not show from where they retrieved these constraints (e.g. Relational.OWL, Ontograte). Some works employed even only physical models (e.g. Project of Buccella et al.).

Another important factor is the expressivity of the ontology language chosen because it specifies the capability of capturing the semantics of data models. Some of the works presented above employed RDF(S) (e.g. MAFRA, RDFGateway) while the others employed the latest standard ontology language OWL (e.g. Project of Buccella et al., Relational.OWL, Ontograte). However, only the project of N. Stojanovic and R. Volz and Ontograte provide a logic foundation for their ontology language. Unfortunately, the logic foundation of both of these two approaches is undecidable. As explained above (Section 3.1.1), not only less expressive than OWL, RDF(S) does not have the underlying inference engine. Even though OWL is employed, the species of OWL is not specified. In other words, users can use OWL Full, which is also undecidable.

Regarding the capability of representing the semantics of data models and of integrity constraints, we see that, except the project of N. Stojanovic and R. Volz, all other works above cannot fully capture the identification constraints that are often modeled as keys in relational database schemas. The project of N. Stojanovic and R. Volz, however, suffers from ambiguous situations that make the transformation be directed by human. Identification constraints are the most basic and important one not only in database modeling but also in the Semantic Web ontologies because all the things in the real world, in some way, need to be identified.

We restate that preserving a maximum of information from data sources under the ontology framework is necessary for the integration. It is important to say that the process of the schema transformation is indispensable. Accordingly, the maturity of current research and practical applications has demonstrated the need for a more formal approach to integrate relational data sources on the Semantic Web. The goal of our approach is to meet this need.

OWL DL is the most well-known and most investigated species of OWL. As the underlying semantics for this current "de facto" standard SW modeling language, DLs have unambiguous (i.e. well-understood declarative) semantics and powerful reasoning algorithms that can automatically classify concepts in hierarchy, memorize the hierarchy for further reasoning and verify a number of properties of an ontology (e.g. correctness, completeness, decidability and complexity). As a result, using OWL DL simplifies the semantic integration problem.

As analyzed in Chapter 2, ORM is an approach that has many advantages over ER and UML in conceptual modeling. It can express the real-world semantics, and can be converted to 5NF relational schemas, which in turn are used to create the physical models of DBs.

Therefore, following semantic annotation approach, we propose employing ORM schemas as data models, and try to translate them into OWL DL ontologies. Source data models will be normalized and lifted to a uniform representation in a Web ontolgogy language so that they remove conficts caused by syntactic heterogeneity. Then the data could be processed by OWL reasoners or be integrated by ontological mapping tool.

To capture the identification constraints, we will extend OWL DL to a Web ontology language called OWL-K[17]. The formal semantics of this language will be provided by a description logic that we call $\mathcal{SHOINK}(D)$. This DL is an extension of the DL underlying OWL-DL with identification constraints. We will introduce OWL-K and its underlying DL language in Chapter 4.

Inference engine for OWL-K will be investigated in Chapter 5, where we introduce a Tableau algorithm for automated reasoning. As a result, the language proposed will be guaranteed decidable, and reasoning services can be applied in data integration systems.

Then, we propose a mechanism to transform ORM schemas into OWL-K without losing the original semantics of data models (cf. Chapter 6). The semantic mapping will be embedded into an automatic tool that will perform the schema transformation (cf. Chapter 6 and Chapter 7).

Table 3.4: Advantages of our proposition over existing works in semantic annotation approach

| | Formal semantics | OWL | Identification constraints | Reasoning support | Automatic transformation | Conceptual data models |
|---|---|---|---|---|---|---|
| MAFRA [98] | Yes | No | No | No | No | No |
| N. Stojanovic and R.Volz [124] | Yes | No | Yes | Undecidable (Flogic) | No | No |
| RDF Gateway [39] | Yes | Yes | No | No | Yes | No |
| Project of Buccella et al. [26] | Yes | Yes | No | No | No | No |
| Relational.OWL [42] | No | Yes | Yes | No | No | No |
| Ontograte [46] | Yes | Yes | No[a] | Undecidable (FOL) | Yes[b] | No |
| Our proposition | Yes | Yes | Yes | Yes | Yes | Yes |

[a]Identification constraint is lost when Web-PDDL is translated into OWL.

[b]Although stating that the transformation is automatic, the authors believed that the interference of users is needed.

The main advantages of our approach over the existing works can be seen in Table 3.4. Our proposition will facilitate the reusability of knowledge[18] that is implicitly inside relational data sources for multi-purpose; leverage exploitations for various

[17]"K" stands for "Key"

[18]The "reusability" of knowledge implies the maximization of using this knowledge among different kinds of applications.

domains and various services on the Semantic Web; and be independent from the run-time characteristics of the underlying sources.

CHAPTER 4

# Web Ontology language OWL-K

Identification constraints (ICs) are very important in applications because all the things in the real world, in some way, need to be identified. Essentially, ICs imply a one-to-one relation between an identifying part and the identified one. Even though this important feature has been modeled in Database schemas under the name *key* a long time ago, as seen in Chapter 3, the current research on data integration in the Semantic Web has not fully captured it yet.

In this chapter, we will study the problem of representing various kinds of identification constraints in OWL DL, the most expressive and decidable species of the current standard Web Ontology language. To know whether OWL DL can be able to represent these identification constraints, an overview of OWL DL is necessary. Therefore, Section 4.1 will be devoted to OWL DL language. Then, in Section 4.2 we investigate the capability of OWL DL in representing ICs. It leads to the requirement that this standard web ontology language should be extended. As a result, in Section 4.3 we introduce a new web ontology language, namely OWL-K. The latter is an extension of OWL DL that allows one to represent ICs in web ontologies. OWL-K, however, could not provide a shared understanding if there is no logic foundation underlying it. Thus, in Section 4.4 we will introduce a new DL language, namely $\mathcal{SHOINK}(\mathbf{D})$, which forms the formal semantics of this language. The work presented in this chapter is a revised and extended version of [107].

## 4.1 Overview of OWL DL Language

OWL DL is designed to provide a language subset that has desirable computational properties for automated reasoning systems. It has the same set of constructors as OWL, but restricts them to be used in a way satisfying decidable inference. OWL DL, therefore, can be considered as a DL for the business segment. The complexity of the ontology entailment problem of OWL DL is NexpTime-complete [82]. Actually, underlying OWL DL is the DL $\mathcal{SHOIN}(\mathbf{D})$, which is an extension of $\mathcal{SHOQ}(\mathbf{D})$ [83] with inverse roles and restricted to unqualified number restrictions. What makes OWL DL a Semantic Web language, however, is not its semantics, which is quite standard for a DL, but its RDF/XML exchange syntax besides an abstract frame-like syntax. In this section, we will overview this language.

### 4.1.1 Data types in OWL DL

OWL DL uses many of the built-in XML Schema data types as its built-in data types, and it adopts the RDF(S) specification of data types and data values.

XML Schema data types [136] are divided into disjoint built-in data types[1] and user-derived data types. Derived data types can be defined by derivation from primitive or existing derived data types by the following three means:

- Derivation by *restriction* on an existing type, so as to limit the number of possible values of the derived type,
- Derivation by *union*, i.e., to allow value from a list of data types,
- Derivation by *list*, i.e., to define the list type of an existing data type.

Note that derivation by list is supported by neither RDF(S) nor OWL.

**Example 4.1.1 (An XML schema User-Derived data type).** *The following is the definition of a user-derived data type (of the base data type* integer*) which restricts values to integers less than 26, using the restriction* maxExclusive*, a so-called* facet *in XML terminology*[2]*.*

[1]Refer to [136] for the full list of XML Schema built-in data types.

[2]Refer to [136] for the full list of XML Schema type facets.

```
<xsd:simpleType name = "lessThan26">
  <xsd:restriction base = "xsd:integer">
    <xsd:maxExclusive value = "26"/>
  </xsd:restriction>
</xsd:simpleType>
```

◇

RDF(S) provides a specification of data types and data values. It allows the use of data types defined by any external type systems, e.g. the XML Schema type system, which conform to this specification.

**Definition 4.1.1 (Data type).** *A data type $d$ is characterized by a lexical space $L(d)$ which is a non-empty set of Unicode strings, a value space $V(d)$ which is a non-empty set of values, and a total mapping $L2V(d)$ from the lexical space to the value space.*

For example, the XML Schema data type xsd:boolean is characterized by $L(\text{xsd:boolean}) = \{$"0", "1", "T", "F"$\}$, $V(\text{xsd:boolean})=\{true, false\}$, and a mapping $L2V(\text{xsd:boolean})= \{<$"T", true$>$, $<$"1", true$>$, $<$"0", false$>$, $<$"F", false$>\}$.

*Typed literals* are of the form "$v$" ^^$d$, where $v$ is a Unicode string, called the *lexical form* of the typed literal, and $d$ is a URI reference of a data type. *Plain literals* have a lexical form and optionally a *language tag*[3].

The denotation of a typed literal is the value mapped from its enclosed Unicode string by the lexical-to-value mapping of the data type associated with its enclosed data type URIref. For example, "1"^^xsd:boolean is a typed literal that represents the boolean value *true*, while "1"^^xsd:integer represents the integer 1. Plain literals, e.g., "1", are considered to denote themselves [71]. Accordingly, we use "data literals" for both typed literals and plain literals to denote data values.

In OWL DL, data types are not classes, and object and data type domains are disjoint with each other. Data types (e.g. the `lessThan26` data type) are different from classes (e.g. YoungPerson) in that:

- data types have fixed extension (e.g., the extension of `lessThan26` is all the integers that are less than 26), while classes could have different interpretations in different models;

[3]Definition of "Language tag" can be found in [4]

- we do not have to provide a logical theory for each data type, nor do we have to worry whether a data type is still correct whenever we extend the ontology language;
- a hybrid reasoner can be implemented by combining a reasoner for classes or individuals with one (or possibly more) data type reasoner(s) that can decide the satisfiability problem of data type constraints (cf. Chapter 5).

Since built-in XML Schema data types and RDF(S) specification of data types are usually not enough in many SW and ontology applications, OWL provides the use of so-called enumerated data types, which are built using data literals. For example, oneOf("0"^^xsd:integer "15"^^xsd:integer "30"^^xsd:integer "40"^^xsd:integer) is an enumerated data type, which denotes the set $\{0, 15, 30, 40\}$. Table 4.1 shows data ranges and their semantics in OWL DL, where $\Delta_{\mathbf{D}}$ is the domain of data values, $.^{\mathbf{D}}$ is a data type interpretation function, and $v, v_1, ..., v_n$ are data literals.

Table 4.1: OWL DL data ranges

| Abstract syntax | DL Syntax | Semantics |
|---|---|---|
| rdfs:Literal | $\top_{\mathbf{D}}$ | $\Delta_{\mathbf{D}}$ |
| Data type URIref $d$ | $d$ | $d^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$ |
| oneOf$(v_1...v_n)$ | $\{v_1, ..., v_n\}$ | $v_1^{\mathbf{D}} \cup ... \cup v_n^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$ |
| $v$ | $v$ | $v^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$, $\sharp\{v^{\mathbf{D}}\} = 1$ |

### 4.1.2 Syntax and Semantics

OWL DL provides an abstract syntax and an RDF/XML syntax, as well as a mapping from the abstract syntax to the RDF/XML syntax [81].

The abstract syntax is heavily influenced by frames in general and by the design of OIL in particular. The frame style of the abstract syntax makes it much easier to read (compared to the RDF/XML syntax), and also easier to understand and to use. Moreover, abstract syntax have a direct correspondence with DL syntax (cf. Table 4.2 and Table 4.3).

OWL DL can form descriptions of classes and properties using the constructors shown in Table 4.2. In this table the first column gives the OWL abstract syntax for the constructors, while the second column gives the standard DL syntax. OWL DL uses these description-forming constructors in axioms and facts that provide information about classes, properties, and individuals, as shown in Table 4.3. In these tables, $A$ is a class URI reference; $C, C_1, ..., C_n$ are class descriptions; $S$ is an object property URI reference; $R, R_1, ..., R_n$ are object property descriptions; $o, o_1, ..., o_n$ are individual URI references; $d$ is a data type URI reference; $U$ is a data type property URI reference; $v$ is a data value; $\sharp$ denotes cardinality; $.^{\mathcal{I}}$ is the interpretation function; $\Delta^{\mathcal{I}}$ is the individual domain and $\Delta_{\mathbf{D}}$ is the domain of data values.

Table 4.2: OWL DL class, property descriptions

| Abstract syntax | DL Syntax | Semantics |
|---|---|---|
| Class(A) | A | $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ |
| Class(owl:Thing) | $\top$ | owl:Thing$^{\mathcal{I}} = \Delta^{\mathcal{I}}$ |
| Class(owl:Nothing) | $\bot$ | owl:Nothing$^{\mathcal{I}} = \emptyset$ |
| intersectionOf$(C_1...C_n)$ | $C_1 \sqcap ... \sqcap C_n$ | $(C_1 \sqcap ... \sqcap C_n)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap ... \cap C_2^{\mathcal{I}}$ |
| unionOf$(C_1...C_n)$ | $C_1 \sqcup ... \sqcup C_2$ | $(C_1 \sqcup ... \sqcup C_n)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup ... \cup C_2^{\mathcal{I}}$ |
| complementOf$(C)$ | $\neg C$ | $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$ |
| oneOf$(o_1...o_n)$ | $o_1 \sqcup ... \sqcup o_n$ | $(\{o_1\} \sqcup ... \sqcup \{o_n\})^{\mathcal{I}} = \{o_1^{\mathcal{I}}, ..., o_n^{\mathcal{I}}\}$ |
| $o$ | $o$ | $\sharp\{o^{\mathcal{I}}\} = 1,\, o^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ |
| restriction(R someValuesFrom(C)) | $\exists R.C$ | $(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ |
| restriction(R allValuesFrom(C)) | $\forall R.C$ | $(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$ |
| restriction(R hasValue(o)) | $\exists R.\{o\}$ | $(\exists R.\{o\})^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \langle x, \{o\}^{\mathcal{I}}\rangle \in R^{\mathcal{I}}\}$ |
| restriction(R minCardinality(n)) | $\geq nR$ | $(\geq nR)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}}\} \geq n\}$ |
| restriction(R maxCardinality(n)) | $\leq nR$ | $(\leq nR)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}}\} \leq n\}$ |
| restriction(U someValuesFrom(d)) | $\exists U.d$ | $(\exists U.d)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y.\langle x, y\rangle \in U^{\mathcal{I}} \wedge y \in d^{\mathbf{D}}\}$ |
| restriction(U allValuesFrom(d)) | $\forall U.d$ | $(\forall U.d)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y.\langle x, y\rangle \in U^{\mathcal{I}} \rightarrow y \in d^{\mathbf{D}}\}$ |
| restriction(U hasValue(v)) | $\exists U.v$ | $(\exists U.v)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \langle x, v^{\mathbf{D}}\rangle \in U^{\mathcal{I}}\}$ |
| restriction(U minCardinality(n)) | $\geq nU$ | $(\geq nU)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp\{y.\langle x, y\rangle \in U^{\mathcal{I}}\} \geq n\}$ |
| restriction(U maxCardinality(n)) | $\leq nU$ | $(\leq nU)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \sharp\{y.\langle x, y\rangle \in U^{\mathcal{I}}\} \leq n\}$ |
| ObjectProperty(S) | $S$ | $S^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ |
| inverseOf(S) | $S^{-}$ | $(S^{-})^{\mathcal{I}} = \{\langle x, y\rangle \mid \langle y, x\rangle \in S^{\mathcal{I}}\}$ |
| DatatypeProperty(U) | $U$ | $U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathbf{D}}$ |

As shown in Table 4.2, OWL classes can be specified as logical combinations (intersections, unions, or complements) of other classes, as enumerations of specified objects. The major extension over RDFS is the ability of OWL DL to provide restrictions on how properties behave locally to a class. A class can be defined with a particular property restricted so that

- all the values for the property in instances of the class must belong to a certain class (or data type);
- at least one value must come from a certain class (or data type);
- there must be at least a specific value;
- there must be at least or at most a certain number of distinct values.

Table 4.3 shows that classes can be organized in a subsumption (subclass) hierarchy, disjointness statements can be made on classes. OWL DL can also declare properties with their domains and ranges, organize them into a subproperty hierarchy. Equivalence statements can be made on classes and on properties. OWL DL can also state whether a property is transitive, symmetric, functional, or inverse of another property. It can express which individuals belong to which classes, and what the property values are of specific individuals. Equality and inequality can be asserted between individuals.

Table 4.3: OWL DL axioms and facts

| Abstract syntax | DL Syntax | Semantics |
|---|---|---|
| Class(A partial $C_1 ... C_n$) | $A \sqsubseteq C_1 \sqcap ... \sqcap C_n$ | $A^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \sqcap ... \sqcap C_n^{\mathcal{I}}$ |
| Class(A complete $C_1 ... C_n$) | $A \equiv C_1 \sqcap ... \sqcap C_n$ | $A^{\mathcal{I}} \equiv C_1^{\mathcal{I}} \sqcap ... \sqcap C_n^{\mathcal{I}}$ |
| EnumeratedClass(A $o_1 ... o_n$) | $A \equiv \{o_1\} \sqcap ... \sqcap \{o_n\}$ | $A^{\mathcal{I}} \equiv \{o_1^{\mathcal{I}}, ..., o_n^{\mathcal{I}}\}$ |
| SubClassOf($C_1, C_2$) | $C_1 \sqsubseteq C_2$ | $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ |
| EquivalentClasses($C_1 ... C_n$) | $C_1 \equiv ... \equiv C_n$ | $C_1^{\mathcal{I}} = ... = C_n^{\mathcal{I}}$ |
| DisjointClasses($C_1 ... C_n$) | $C_i \sqsubseteq \neg C_j$, $(1 \le i < j \le n)$ | $C_i^{\mathcal{I}} \cap C_j^{\mathcal{I}} = \emptyset$, $(1 \le i < j \le n)$ |
| DatatypeProperty($U$ super($U_1$) ... super($U_n$) | $U \sqsubseteq U_i, (1 \le i \le n)$ | $U^{\mathcal{I}} \subseteq U_i^{\mathcal{I}}, (1 \le i \le n)$ |
| domain($C_1$) ... domain($C_m$) | $\ge 1U \sqsubseteq C_i, (1 \le i \le m)$ | $U^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta_{\mathbf{D}}{}^{\mathcal{I}}, (1 \le i \le m)$ |
| range($d_1$) ... range($d_k$) | $\top \sqsubseteq \forall U.d_i, (1 \le i \le k)$ | $U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times d_i^{\mathcal{I}}, (1 \le i \le k)$ |
| [Functional]) | $\top \sqsubseteq \le 1U$ | $\{\langle x, y\rangle \mid \sharp\{y.\langle x, y\rangle \in U^{\mathcal{I}}\} \le 1 \forall x \in \Delta^{\mathcal{I}}\}$ |
| SubPropertyOf($U_1\, U_2$) | $U_1 \sqsubseteq U_2$ | $U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$ |
| EquivalentProperties($U_1 ... U_n$) | $U_1 \equiv ... \equiv U_n$ | $U_1^{\mathcal{I}} = ... = U_n^{\mathcal{I}}$ |
| ObjectProperty($R$ super($R_1$) ... super($R_n$) | $R \sqsubseteq R_i, (1 \le i \le n)$ | $R^{\mathcal{I}} \subseteq R_i^{\mathcal{I}}, (1 \le i \le n)$ |
| domain($C_1$) ... domain($C_m$) | $\ge 1R \sqsubseteq C_i, (1 \le i \le m)$ | $R^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta^{\mathcal{I}}, (1 \le i \le m)$ |
| range($C_1$) ... range($C_k$) | $\top \sqsubseteq \forall R.C_i, (1 \le i \le k)$ | $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_i^{\mathcal{I}}, (1 \le i \le k)$ |
| [inverseOf(S)] | $R \equiv S^-$ | $R^{\mathcal{I}} = (S^-)^{\mathcal{I}}$ |
| [Symmetric] | $R \equiv R^-$ | $R^{\mathcal{I}} = (R^-)^{\mathcal{I}}$ |
| [Functional] | $\top \sqsubseteq \le 1R$ | $\{\langle x, y\rangle \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}}\} \le 1 \forall x \in \Delta^{\mathcal{I}}\}$ |
| [InverseFunctional] | $\top \sqsubseteq \le 1R^-$ | $\{\langle x, y\rangle \mid \sharp\{y.\langle x, y\rangle \in (R^-)^{\mathcal{I}}\} \le 1 \forall x \in \Delta^{\mathcal{I}}\}$ |
| [Transitive]) | Trans(R) | $R^{\mathcal{I}} = (R^{\mathcal{I}})^+$ |
| SubPropertyOf($R_1, R_2$) | $R_1 \sqsubseteq R_2$ | $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$ |
| EquivalentProperties($R_1 ... R_n$) | $R_1 \equiv ... \equiv R_n$ | $R_1^{\mathcal{I}} = ... = R_n^{\mathcal{I}}$ |
| AnnotationProperty(R) | | |
| Individual($o$ | | |
| type($C_1$) ... type($C_n$) | $o \in C_i$ | $o^{\mathcal{I}} \in C_i^{\mathcal{I}}$ |
| | $\langle o, o_i\rangle \in R_i$ | $\langle o^{\mathcal{I}}, o_i^{\mathbf{D}}\rangle \in R_i^{\mathcal{I}}$ |
| value($R_1\, o_1$)...value($R_n\, o_n$)) | | |
| | $\langle o, v_i\rangle \in U_i$ | $\langle o^{\mathcal{I}}, v_i^{\mathbf{D}}\rangle \in R_i^{\mathcal{I}}$ |
| value($U_1\, v_1$)...value($U_n\, v_n$)) | | |
| SameIndividual($o_1 ... o_n$) | $o_1 = ... = o_n$ | $o_1^{\mathcal{I}} = ... = o_n^{\mathcal{I}}$ |
| DifferentIndividuals($o_1 ... o_n$) | $o_i \ne o_j$ | $o_i^{\mathcal{I}} \ne o_j^{\mathcal{I}}$ |

OWL properties can be either data type properties or object properties. The former relates objects to data values in data ranges, while the latter relates objects to objects. The following four property characteristics can never be specified for data type properties:

- inverse of,

- inverse functional,
- symmetric, and
- transitive.

The semantics of OWL DL includes also some unusual (w.r.t DL) aspects. Annotations are given a simple separate meaning that can be used to associate information with classes, properties, and individuals in a manner compatible with the RDF semantics.

An example of OWL DL ontology in abstract syntax can be seen in Figure 4.1, which illustrates the simple Dormitory ontology presented in Example 3.2.1. Full details on the abstract syntax of OWL DL can be found in [81].

```
 Ontology(
 Annotation(rdfs:label "Ontology of Dormitory")
 Annotation(rdfs:comment "An OWL ontology example")

 Class(Certificate partial)
 Class(Student partial)

 ObjectProperty(hasTrainingCertificate
  domain(Student)
  range(Certificate))

 Class(Trainee partial
  restriction(hasTrainingCertificate allValuesFrom(Certificate))
  Student)

 ObjectProperty(compatriot
  domain(Student)
  range(Student)
  Transitive)
)
```

Figure 4.1: An example ontology in the OWL DL abstract syntax

The OWL RDF/XML syntax is the exchange syntax for OWL DL. RDF is known to have restrictions to provide a well-formed syntax for OWL (cf. section 2.2). Therefore, the RDF/XML syntax form of an OWL DL ontology is valid if and only if it can be translated (according to the mapping rules provided in [81]) from the abstract syntax form of the ontology. An entity of an OWL DL ontology can be a class, a property, or an instance. All entities in an OWL DL ontology are resources, i.e. elements of the RDF domain of discourse, and can be identified by URIs. Classes and individuals can be named or not. In the former case, they are referred to by URIs and their names are these URIs. In the latter case, they are called "blank nodes" and are not be referenced by any URI. Entity descriptions in OWL DL ontologies can be represented in the form of RDF triplets whose predicates are OWL primitives. The latter are properties predefined in OWL or in RDF(S), such as rdfs:subClassOf and owl:equivalentClass (SubClassOf and EquivalentClasses in abstract syntax respectively). The full list of OWL primitives can be found in [114]. All the entities in a legal OWL ontology are typed by the primitive rdf:type. That is, there exists at least a triplet whose subject, predicate and object are the entity concerned, rdf:type, and an OWL class respectively. Individuals in an ontology are typed by the classes described in that ontology. The classes and properties are typed by predefined classes in OWL. The full list of OWL classes can be found in [114].

OWL DL direct model-theoretic semantics is very similar to the semantics provided by DLs, except that symbols for classes and properties, etc. are URI references instead of the usual names (strings), and that the model-theoretic semantics includes semantics for annotation properties and ontology properties (cf. [81]). Consequently, we do not investigate this semantic formalism in this thesis.

## 4.2 OWL DL with Identification Constraints

In this section, we will review the capability of OWL DL in describing Identification constraints, show its limitations and identify the requirements that an extension of OWL DL must satisfy.

In the sense of OWL DL, ICs can be defined as to state that a certain set of properties uniquely identifies the instances of a given class. Essentially, these constraints put a one-to-one relation between sets of values of respective properties and instances of a class. Although supporting considerable expressive power to the Semantic Web (cf. Section 4.1), the mechanism to express this kind of constraints in OWL DL is still very seriously limited. In particular, OWL DL provides two constructors, namely (in abstract syntax) Functional and InverseFunctional, which can be used to link individuals together. The semantics of Functional and InverseFunctional can be seen in Table 4.3. However, we restate it here for easier reference:

- If a property, P, is tagged as Functional then for all x, y, and z: P(x,y) and P(x,z) implies y = z. For example, if the property hasFlag is characterized

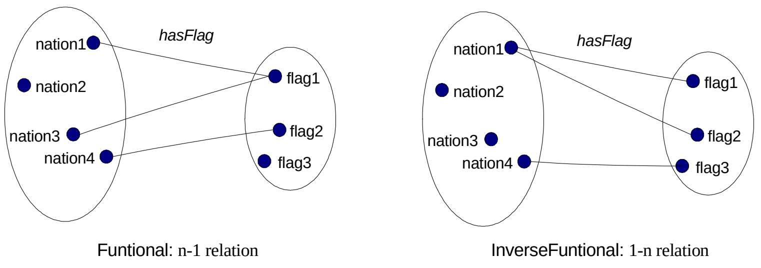


Figure 4.2: Examples showing the limitations of OWL DL in representing ICs

as Functional, then each nation has at most one flag; if a nation has "two flags", they must be the same (see the illustration of Functional constructor in Figure 4.2).

- If a property, P, is tagged as InverseFunctional then for all x, y and z: P(y,x) and P(z,x) implies y = z. For example, if the property hasFlag above is characterized as InverseFunctional, then two nations could be inferred to be identical based on having the same flag (see the illustration of InverseFunctional in Figure 4.2).

Suppose that one would like to identify uniquely each nation by its flag. The constructor Functional shown above obviously does not respond to this requirement while InverseFunctional seems to fit well. Accordingly, one can think of InverseFunctional as defining ICs.

However, Figure 4.2 shows that InverseFunctional does not require that two separate flags must correspond to separate nations. Consequently, one nation can be identified by different flags, which is not expected. Actually, the relation put by InverseFunctional is a one-to-many relation whereas Functional represents many-to-one relations. Therefore one can think of using both of these constructors to represent the notion of ICs. Applying both these constructors on the property hasFlag, one nation cannot be identified by different flags anymore.

Nevertheless, the relations put by Functional and InverseFunctional are not compulsory to all elements of the domain. As shown in Figure 4.2, not all nations must have flags and vice versa. In the sense of DB schemas, we can say that these constructors allow for expressing null keys, which is never permitted in primary keys. To enforce a compulsory participation, one can think of applying exists restriction to both properties hasFlag and inverseOf(hasFlag): restriction(hasFlag someValuesFrom(Flag)), restriction(inverseOf(hasFlag) someValuesFrom(Nation)).

However, if one would like to specify that "Cities are uniquely identified by City codes", an ontology should constrain all instances of the class City to have distinct values for their city codes. This requirement, unfortunately, cannot be satisfied by any combination of the three constructors presented above. The reason is that in OWL DL InverseFunctional cannot be applied to data type property (cf. Section 4.1). Consequently, it is impossible in OWL DL to assert that hasCitycode, which has the range of data type *integer*, is an identifier for instances of the class City.

Furthermore, the DL syntax and semantics of InverseFunctional and Functional show that these constructors are special kinds of number restrictions (cf. Table 4.3 and 4.2). They essentially apply the restrictions to all elements of the universe, i.e. owl:Thing - so called *global* restrictions. That is why one cannot use these constructors to put dependencies on certain classes. For example, suppose that the classes Nation and NationHistory both share the property hasFlag. One would like to assert that hasFlag is a unique identifier for instances of Nation, but not for instances of NationHistory. InverseFunctional cannot express this.

In addition, these constructors are designed to put only relations between two elements (only possibly referring to simple keys in DB schemas), while those between an element and a set of elements (corresponding to compound keys in DB schemas) cannot be expressed. Let us revisit the class NationHistory mentioned above in Example 4.2.1.

**Example 4.2.1.** *One would like to state that instances of class NationHistory are uniquely identified by a couple of properties (hasFlag, onDate), where hasFlag is an ObjectProperty whose values are flags and onDate is a DatatypeProperty whose values are dates.*

The constraint described in this example is obviously out of the expressivity of OWL DL. The language has no constructors to describe that an "instance" of the couple (hasFlag, onDate) uniquely identifies an instance of the class NationHistory. Even if one tries to express an IC concerning only objects, i.e., suppose that onDate is an object property, by using the combination of InverseFunctional and Functional, it cannot capture the case of two nation history items sharing the same flag on different dates. It is even impossible to describe this constraint with any combination of constructors in OWL DL.

As a result, to express ICs in the Semantic Web, a new mechanism must be designed that needs to satisfy the following requirements:

1. It should provide the "real" ICs, i.e., one-to-one relations;
2. It should support ICs on specific classes, or so called non-global ICs;

3. It should provide ICs for data type properties, object properties, and sets of these two kinds of properties;
4. It should support both kinds of constraints corresponding to simple and compound keys in DB schemas;
5. It should be layered on top of OWL DL. The extension of OWL DL should not influence the existing syntax of OWL DL. The users of OWL are therefore able to create their web applications or ontologies in the extended language without being influenced by the features designed for data integration. Although the IC utility is useful not only in data integration but also in other applications, users can switch off this utility if they do not need it;
6. The extension with this mechanism should have formal semantics;
7. The extension with this mechanism should be a *decidable* extension of OWL DL.

The following sections and Chapter 5 will introduce a such mechanism that makes an extended language of OWL DL called OWL-K.

## 4.3 Modeling ICs in OWL-K

In this section, we introduce a mechanism to represent ICs into OWL DL, which creates an extension language called OWL-K ("K" stands for "key"). The new mechanism is designed separately from the existing syntax of OWL DL so that it can be switched off if the users do not need it.

### 4.3.1 Vocabulary

In order to satisfy the requirements shown in Section 4.2, we extend OWL DL with *IC assertions* resulting in the language OWL-K. In this language, IC assertions are modeled as entities. They are neither classes nor properties. Hence IC assertions are defined as instances of a new class owlk:ICAssertion, which is a subclass of rdfs:Resource (cf. Figure 4.3).

Figure 4.3 shows information about the class hierarchy using a "nodes and arcs" graph representation of the RDF data model. If a class is a subset of another, then there is an rdfs:subClassOf (s) arc from the node representing the former class to

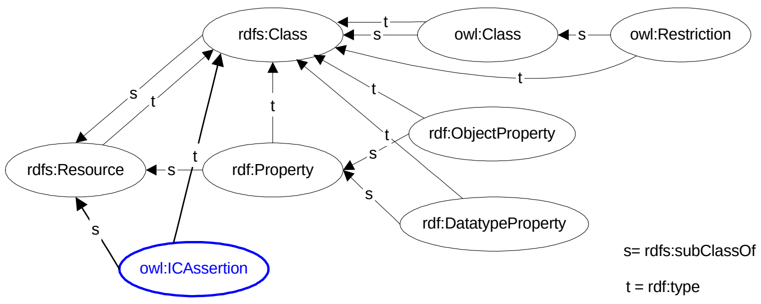


Figure 4.3: Class Hierarchy for OWL-K

the node representing the latter. Similarly, if a resource is an instance of a class, then there is an rdf:type (t) arc from the resource to the node representing the class. Note that not all resources, classes and arcs are shown. We only show the principle resources, classes and arcs relating to our discussion for the extension. The rest is the same as for OWL DL [114].

As specified in the requirements (cf. Section 4.2), an IC must put a constraint on a set of properties. Let us see how OWL DL provides property restrictions:

- First, property restrictions are defined as classes while ICs do not create new classes;

- Second, OWL DL distinguishes two kinds of property restrictions: value constraints and cardinality constraints. *Value constraints* put constraints on the range of *a* property when applied to this particular class description. *Cardinality constraints* put constraints on the number of values *a* property can take, in the context of this particular class description.

So restriction constructors in OWL DL put restrictions on only *one* property while an IC puts a restriction on *a collection* of properties. Restriction constructors in OWL DL, therefore, do not agree with ICs. To express IC assertions, we introduce new kinds of restriction constructors, namely owlk:onClass and owlk:byProperty.

owlk:onClass is used to specify the class an IC is applied on. owlk:byProperty is used to specify the property in the collection of properties identifying instances of a given class. Since ICs can be applied both to data type properties and object properties, `owlk:byProperty` is designed to have range both of the data-valued and object properties. Table 4.4 shows the vocabulary extension of OWL-K compared with OWL DL.

Table 4.4: Vocabulary extension for OWL-K

| rdfs:Class | rdfs:subClassOf |
|---|---|
| owlk:ICAssertion | rdfs:Resource |

| Property name | Type | rdfs:domain | rdfs:range |
|---|---|---|---|
| owlk:onClass | rdf:ObjectProperty | owlk:ICAssertion | owl:Class |
| owlk:byProperty | rfd:Property | owlk:ICAssertion | owl:Property |

## 4.3.2 Abstract Syntax

The abstract syntax is used to facilitate access to and evaluation of the language. It is specified by means of a version of Extended BNF:

- terminals are quoted; non-terminals are bold and not quoted,
- alternatives are either separated by vertical bars (|) or are given in different productions,
- components that can occur at most once are enclosed in square brackets ([...]); components that can occur any number of times (including zero) are enclosed in braces ({...}). Whitespace is ignored.

IC assertions in OWL-K ontologies must be identifiable and referable like any other entity of ontologies. Hence, as for classes, properties and instances in OWL DL ontologies, we associate with each IC assertion in an OWL-K ontology an identifier, which is a URI reference.

As the axioms modeled in OWL DL, a new kind of axioms to express IC assertions is added as follows:

```
          ICAssertionID  ::= URIreference
                  axiom  ::= 'ICAssertion(' ICAssertionID
                                  description
                                  propertyID {propertyID}')'
             propertyID  ::= datavaluedPropertyID |
                                  individualvaluedPropertyID
   datavaluedPropertyID  ::= URIreference
individualvaluedPropertyID ::= URIreference
```

**description** is defined as in the OWL DL abstract syntax, which can be a class identifier, restrictions, boolean combination of other descriptions, or also a set of individuals.

With this syntax, the IC in Example 4.2.1 can be described in the abstract syntax as follows:

```
ICAssertion(NationHistoryIC NationHistory hasFlag onDate)
```

### 4.3.3 RDF Graphs

In the Semantic Web environment, an OWL ontology is in fact an RDF graph, which is in turn a set of RDF triples. Hence it is necessary to relate specific abstract syntax ontologies with specific RDF/XML documents and their corresponding graphs. We provide a mapping from the abstract syntax for OWL-K to the exchange syntax, namely RDF/XML syntax. Since OWL-K is the extension of OWL DL, it inherits the existing mapping for OWL DL (see section 4 of [81]). We introduce here only the mapping for IC assertions (cf. Table 4.5, where "T" stands for "triples"). So that our extension preserves the normative relationship between the abstract syntax and the exchange syntax.

Table 4.5: Transformation of IC assertion to triples

| Abstract syntax S | Transformation - T(S) |
|---|---|
| ICAssertion($ICAssertionID$ $description$ $propertyID_1$ $\ldots propertyID_n$) | $ICAssertionID$ rdf:type owlk:ICAssertion.<br>$ICAssertionID$ rdf:type rdfs:Resource.[opt]<br>$ICAssertionID$ owlk:onClass T($description$).<br>$ICAssertionID$ owlk:byProperty T($propertyID_1$).<br>...<br>$ICAssertionID$ owlk:byProperty T($propertyID_n$). |

As a result, the IC in example 1 is represented in the exchange syntax as follows:

```
<owlk:ICAssertion rdf:ID = "NationHistoryIC">
    <owlk:onClass rdf:resource = "#NationHistory" />
    <owlk:byProperty rdf:resource = "#hasFlag"/>
    <owlk:byProperty rdf:resource = "#onDate"/>
</owlk:ICAssertion>
```

## 4.4 $\mathcal{SHOINK}(\mathbf{D})$ - Logic Foundation of OWL-K

In this section, we introduce the logic foundation of OWL-K, namely a DL $\mathcal{SHOINK}(\mathbf{D})$, which forms the formal semantics of our new Web ontology language. While in OWL-K we talk about *classes*, *object properties* and *data type* *properties*, in $\mathcal{SHOINK}(\mathbf{D})$ we call them *concepts*, *abstract roles* and *concrete roles* respectively. In DLs, we call individual domain the (abstract) interpretation domain, and domain of data values the *concrete interpretation domain.* An OWL-K ontology can be seen as an $\mathcal{SHOINK}(\mathbf{D})$ knowledge base.

Particularly, $\mathcal{SHOINK}(\mathbf{D})$ is an extension of $\mathcal{SHOIN}(\mathbf{D})$, the underlying of OWL DL, with ICs. These constraints are not a standard feature of DLs. In consequence, adding ICs to a particular DL language is always a challenge. Over the past years there have been many efforts to deal with this problem. Before introducing our extension language, we see how those works represent ICs in DLs.

### 4.4.1 Representation of Identification Constraints in DLs

Over the last decade, there have been many efforts to represent ICs, or key constraints in the Database sense, in DLs. One approach is introducing them as new concepts [41, 23, 92]. De Giacomo and colleagues [41] introduced a concept constructor $\chi$ to denote a key constraint for a subset of individuals of a concept $C$. They introduced a description of the form

$$\chi(C, R_1, ..., R_m),$$

where each $R_i$ corresponds to a functional role[4] in our terminology. This description defines a concept $D$ subsumed by $C$ such that no distinct $s, s' \in D$ have the same participation in $R_1, ..., R_m$. However, by this representation, a description $\chi(C, R_1, ..., R_m)$ can yield non-deterministically $D$, i.e., several subsets can result from one description.

Borgida and Weddell [23] overcame this shortcoming by proposing a definition using the **key** constructor as follows:

$$(\mathbf{key}\, C\, \{R_1...R_m\})$$

with

$$(\mathbf{key}\, C\, \{R_1...R_m\})^{\mathcal{I}} := \{x | x \in \Delta^{\mathcal{I}} \wedge \forall y \in C^{\mathcal{I}} : [R_1^{\mathcal{I}}(x) = R_1^{\mathcal{I}}(y) \wedge ... \wedge R_m^{\mathcal{I}}(x) = R_m^{\mathcal{I}}(y)] \rightarrow x = y\},$$

[4]A role $R$ is called a functional role iff $\top \sqsubseteq \leq 1R$.

where $C$ is a concept, $R_1, ..., R_m$ correspond to functional roles in our terminology, and $R_i^{\mathcal{I}}(x)$ denotes an individual $x_i$ such that $\langle x, x_i \rangle \in R_i^{\mathcal{I}}$.

Under this definition, $(\mathbf{key}\, \bot \, \{R_1...R_m\}) = \top$ for any $\{R_1...R_m\}$. This definition was even later extended to support a kind of roles called *feature paths* [92]. It is obvious that by this definition, a set of attributes $\{R_1...R_m\}$ is neither a key for instances of a concept $C$, which is used in the constraint definition, nor a key for instances of a concept $D$, which is defined by definition. Consequently, the meaning of key constraints is not captured properly. Users must know that if they want to define $\{R_1...R_m\}$ as a key to uniquely identify instances of a concept $C$, they must define a subsumption $C \sqsubseteq \ (\mathbf{key} \quad C \, \{R_1...R_m\})$.

By creating new concepts, this approach needs to define concepts for key so that they fully capture key properties. For example, let $K$ and $K'$ be two concepts generated by setting two key constraints on a concept $C$. The only difference between the constraints for $K$ and $K'$ is that the set of key attributes for $K$ is subsumed by the one for $K'$. In the relational database sense, the relation between $K$ and $K'$ must be that $K$ subsumes $K'$. The definition for key constraints in [41] does not capture this property. Moreover, the DL languages in the works following this approach are limited in expressivity or/and in decidability.

An other approach that avoids the above limitations of key representation is to declare key constraints. Calvanese et al. [28] proposed a *key-declaration* in the form:

$$\underline{\text{key}} \quad R_1, ..., R_m,$$

where $R_i$ corresponds to (possibly non-functional) roles in our terminology. Note that there is no concept name in a key-declaration. A class $C$ satisfies a key-declaration "$\underline{\text{key}} \quad R_1, ..., R_m$" if $\forall o \in C^{\mathcal{I}}$ there are $o_1, ..., o_m \in \Delta^{\mathcal{I}}$ such that $\langle o, o_i \rangle \in R_i^{\mathcal{I}}$ for $1 \leq i \leq m$, and there is no other $o' \in C^{\mathcal{I}}$ such that $o' \neq o$ for which these conditions hold. The semantic definition of this declaration exposes that their key constraints are *not* one-to-one relations, i.e., one instance of $C$ can have more than one participation in $R_1, ..., R_m$. Actually, key-declarations are allowed only in class definitions, where key constraints are assigned to concepts.

In a later work that extends $\mathcal{DLR}$, a DL with n-ary roles [29], Calvanese and his colleagues put concepts into definitions of key constraint and called these *identification assertions*. An identification assertion on a concept $C$ has the form:

$$(\mathbf{id} \quad C \ [i_1]R_1, ..., [i_m]R_m),$$

where each $R_j$ is an n-ary role, and $[i_j]R_j$ denotes the component $i_j$ of $R_j$. It is obvious that this representation has the similar form to the one in [23]. However, its semantics is totally different. An interpretation $\mathcal{I}$ satisfies the assertion (**id** $C$ $[i_1]R_1, ..., [i_m]R_m$) if, for all $a, b \in C^{\mathcal{I}}$ and for all $t_1, s_1 \in R_1^{\mathcal{I}}, ..., t_m, s_m \in R_m^{\mathcal{I}}$, we have that:

$$\left.\begin{array}{l} a = t_1[i_1] = ... = t_m[i_m], \\ b = s_1[i_1] = ... = s_m[i_m], \\ t_j[i] = s_j[i],\ for\, j \in \{1, ..., m\},\ and\, for\, i \neq i_j \end{array}\right\} implies \quad a = b.$$

The authors defined a box called $\mathcal{F}$ that contains all identification assertions. A knowledge base consists of a TBox, an ABox and a box $\mathcal{F}$. Considering the semantics of key assertions, we find that key assertions, actually, do not capture properly the meaning of key constraints in relational database neither, i.e. one-to-one relations, but one-to-n relations. Furthermore, there is no known practical decision procedure to solve the problem of decidability in $\mathcal{DLR}$. Even if such a procedure exists, it causes some problems of tractability [85].

This approach is also followed by C. Lutz et al. [97]. They add key constraints to the DLs with the concrete domain $\mathcal{ALCO}(\mathcal{D})$ and $\mathcal{SHOQ}(\mathcal{D})$, resulting in two extended languages $\mathcal{ALCOK}(\mathcal{D})$ and $\mathcal{SHOQK}(\mathcal{D})$ respectively. The authors use also a box distinct from ABox and TBox to store key assertions. Formally, they introduce key boxes that are sets of key assertions of the form

$$(u_1, ..., u_m\ keyfor\ C),$$

where $m \geq 1$, each $u_i$ is called a *path* which is a composition $f_1...f_n g$ of $n$ abstract functional roles $f_1, ..., f_n$ $(n \geq 0)$ and a concrete functional role $g$, and $C$ is a concept. For $d \in \Delta^{\mathcal{I}}$, $u_i^{\mathcal{I}}(d)$ is defined as $g^{\mathcal{I}}(f_n^{\mathcal{I}}...(f_1^{\mathcal{I}}(d))...)$.

The interpretation $\mathcal{I}$ satisfies a key assertion $(u_1, ..., u_k\ keyfor\ C)$ if, for any $a, b \in C^{\mathcal{I}}$,

$$u_1^{\mathcal{I}}(a) = u_1^{\mathcal{I}}(b), ..., u_n^{\mathcal{I}}(a) = u_n^{\mathcal{I}}(b) \text{ implies } a = b.$$

This semantics respects the one-to-one relation of ICs. However, note that the authors use concrete functional roles which restrict the key to only a set of values to obtain this objective. This, as a consequence, does not allow to describe all the possibilities of the constraint as shown in Section 4.2.

Besides, even though Lutz et al. [97] add ICs to DLs with expressive concrete domains (i.e. domains with $n$-ary predicates), restricted by concrete functional role, $\mathcal{ALCOK}(\mathcal{D})$ and $\mathcal{SHOQK}(\mathcal{D})$ do not permit some concrete constructors (e.g. $\forall U.d, \leq nU.d, \geq nU.d$) that are allowed in OWL. The concrete domain in $\mathcal{SHOIQK}(\mathbf{D})$, which will be introduced in the next section, is less expressive. Nevertheless, it plays a prominent role in the construction of ontologies.

### 4.4.2 Universal Concrete domain

A major concern when providing DL reasoning services for OWL is the support for OWL datatyping. *Concrete data types* are defined by external type systems (e.g. XML schema type system) to represent literal values.

A set of concrete data types (or data types for short) composes a *universal concrete domain* $\mathbf{D}$, in which the predicates representing data types are unary. For example, with the data type $\geq_{26}$ defined in $\mathbf{D}$ as a set of *integer* values greater than or equal to 26, we can describe the concept "people who are at least 26 years old" as Person $\sqcap \exists age. \geq_{26}$ where Person is a concept and *age* is a *concrete role.* Formally, a universal concrete domain $\mathbf{D}$ is defined in Definition 4.4.1 (presented slightly differently from the original version).

**Definition 4.4.1 (Universal concrete Domain [112]).** *A universal concrete domain $\mathbf{D}$ is a pair $(\Delta_{\mathbf{D}}, \Phi_{\mathbf{D}})$, where $\Phi_D$ is a set of concrete data type (unary predicate) names and $\Delta_{\mathbf{D}}$ is the domain of all these data types. Each data type name $d \in \Phi_{\mathbf{D}}$ is associated with a set $d^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$. $\neg d$ is the name for the negation of a data type name $d$ with $(\neg d)^{\mathbf{D}} = \Delta_{\mathbf{D}} \backslash d^{\mathbf{D}}$.*

*Let $\boldsymbol{V}$ be a set of variables. A data type conjunction of the form*

$$c = \bigwedge_{j=1}^{k} d_j(v_j), \tag{4.1}$$

*where $d_j \in \Phi_{\mathbf{D}}$ (possibly negated) and $v_j \in \boldsymbol{V}$, is called* satisfiable *iff there exists a function $\delta$ mapping the variables in $c$ to data values in $\Delta_{\mathbf{D}}$ such that $\delta(v_j) \in d_j^{\mathbf{D}}$ for $1 \leq j \leq k$. Such a function $\delta$ is called a* solution *for c.*

*A set of data types $\Phi_{\mathbf{D}}$ is called* conforming *iff the satisfiability problem for finite (possibly negated) data type conjunctions over $\Phi_{\mathbf{D}}$ is decidable.*

For example, let $N = (<_2^{\mathbf{D}}, \{<_2\})$ be a universal concrete domain where the domain of the data type $<_2$ is $<_2^{\mathbf{D}} = \{0, 1\}$. A data type conjunction for the concept $\geq 3U \sqcap \forall U. <_2$, where $U$ is a concrete role, should be $c = <_2 (v_1) \wedge <_2 (v_2) \wedge <_2 (v_3)$, where $\{v_1, v_2, v_3\} \in \mathbf{V}$. $c$ is satisfiable if and only if there exists a solution $\delta$ such that $\delta(v_1) \in <^{\mathbf{D}}_2$, $\delta(v_2) \in <^{\mathbf{D}}_2$ and $\delta(v_3) \in <^{\mathbf{D}}_2$.

### 4.4.3 $\mathcal{SHOINK}(\mathbf{D})$ Syntax and Semantics

**Definition 4.4.2.** *Let $\mathbf{R}_+$ be a set of transitive role names, $\mathbf{R}$ be a set of abstract role names with $\mathbf{R}_+ \subseteq \mathbf{R}$. A set of abstract roles $\mathbf{R}_A$ is $\mathbf{R} \cup \{R^- | R \in \mathbf{R}\}$ where $R^-$ is the inverse role of $R$. Let $\boldsymbol{R}_\mathbf{D}$ be a set of concrete role names, $\boldsymbol{R}_A \cap \boldsymbol{R}_\mathbf{D} = \emptyset$.*

*The set of $\mathcal{SHOINK}(\mathbf{D})$-*roles *(or* roles *for short) is $\mathbf{R}_A \cup \mathbf{R}_\mathbf{D}$.*

*A* role hierarchy *(*Rbox*) $\mathcal{R}$ is a finite set of role inclusion axioms which are of the form $R \sqsubseteq S$ for $R, S \in \boldsymbol{R}_A$ or $R, S \in \boldsymbol{R}_\mathbf{D}$.*

*An interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ consists of a non-empty abstract interpretation domain $\Delta^\mathcal{I}$, $\Delta^\mathcal{I} \cap \Delta_\mathbf{D} = \emptyset$, and a function $\cdot^\mathcal{I}$ that maps every role to a subset of $\Delta^\mathcal{I} \times \Delta_\mathbf{D}$ or of $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$ such that for $P \in \mathbf{R}$ and $R \in \mathbf{R}_+$,*

$$\langle x, y\rangle \in P^\mathcal{I} \text{ iff } \langle y, x\rangle \in (P^-)^\mathcal{I},$$
$$\text{if } \langle x, y\rangle \in R^\mathcal{I} \text{ and } \langle y, z\rangle \in R^\mathcal{I} \text{ then } \langle x, z\rangle \in R^\mathcal{I}.$$

*The interpretation $\mathcal{I}$ satisfies a role hierarchy $\mathcal{R}$ iff $R^\mathcal{I} \subseteq S^\mathcal{I}$ for each $R \sqsubseteq S \in \mathcal{R}$. Such an interpretation is called a model of $\mathcal{R}$.*

Table 4.6: $\mathcal{SHOINK}(\mathbf{D})$ role syntax and semantics

| Constructor name | Syntax | Semantics |
|---|---|---|
| abstract atomic role | $R$ | $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ |
| concrete role | $U$ | $U^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta_\mathbf{D}$ |
| transitive role | $R \in \mathbf{R}_+$ | $R^\mathcal{I} = R^{\mathcal{I}^+}$ |
| inverse role | $R^-$ | $\{\langle x, y\rangle \| \langle y, x\rangle \in R^\mathcal{I}\}$ |
| role hierarchy | $R \sqsubseteq S$ | $R^\mathcal{I} \subseteq S^\mathcal{I}$ |

The $\mathcal{SHOINK}(\mathbf{D})$ role syntax and semantics can be seen in Table 4.6. Besides, there are some notations for roles:

1. To avoid roles in the form of $R^{--}$, a function Inv is defined to return the inverse of a role. For $R \in \mathbf{R}$, $\text{Inv}(R) := R^-$ and $\text{Inv}(R^-) := R$.

2. Let $\overset{*}{\sqsubseteq}_{\mathcal{R}}$ be the transitive-reflexive closure of $\sqsubseteq$ on $\mathcal{R} \cup \{\text{Inv}(R) \sqsubseteq \text{Inv}(S) | R \sqsubseteq S \in \mathbf{R}_A\}$, the equivalence of two roles $R \equiv_{\mathcal{R}} S$ means that $R \overset{*}{\sqsubseteq}_{\mathcal{R}} S$ and $S \overset{*}{\sqsubseteq}_{\mathcal{R}} R$.

3. A role $R$ is transitive if and only if its inverse $\text{Inv}(R)$ is transitive. In case $R \equiv_{\mathcal{R}} S$, $S$ is transitive if $R$ or $\text{Inv}(R)$ is a transitive role name. In order to avoid these case distinctions, the function Trans returns true iff $R$ is a transitive role, regardless whether it is a role name, the inverse of a role name, or equivalent to a transitive role name (or its inverse):

   - $\text{Trans}(S, \mathcal{R}) :=$ true if, for some $R$ with $R \equiv_{\mathcal{R}} S$, $R \in \mathbf{R}_+$ or $\text{Inv}(R) \in \mathbf{R}_+$;
   - $\text{Trans}(S, \mathcal{R}) :=$ false otherwise.

4. A role $R$ is a *simple role* w.r.t $\mathcal{R}$ if $\text{Trans}(S, \mathcal{R}) =$ false for all $S \overset{*}{\sqsubseteq}_{\mathcal{R}} R$.

5. In the following, if $\mathcal{R}$ is clear from the context, we may abuse our notation and use $\overset{*}{\sqsubseteq}$, $\equiv$ and $\text{Trans}(S)$ instead of $\overset{*}{\sqsubseteq}_{\mathcal{R}}$, $\equiv_{\mathcal{R}}$ and $\text{Trans}(S, \mathcal{R})$.

**Definition 4.4.3.** *Let $N_C$ be a set of concept names, $N_I$ be a set of nominals with $N_I \subseteq N_C$. The set of $\mathcal{SHOINK}(\mathbf{D})$*-concepts *(or* concepts *for short) is the smallest set such that*

- *every concept name $C \in N_C$ is a concept,*
- *if $C$ and $D$ are concepts and $R$ is an abstract role, $U$ is a concrete role and $d$ is a datatype then $(C \sqcup D)$, $(C \sqcap D)$, $(\neg C)$, $(\forall R.C)$, $(\exists R.C)$, $(\forall U.d)$, and $(\exists U.d)$ are also concepts, and*
- *if $C$ is a concept, $R$ is a simple role*[5]*, or a concrete role and $n \in \mathbb{N}$, then $(\leq nR)$ and $(\geq nR)$ are also concepts.*

*The interpretation function $\cdot^{\mathcal{I}}$ of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ maps every concept to a subset of $\Delta^{\mathcal{I}}$ as shown in Table 4.7.*

**Definition 4.4.4.** *An identification constraint is an expression in the form of*

$$R_1, ..., R_n \, \mathbf{Idfor} \, C,$$

[5]Restricting number restrictions to simple roles is required in order to yield a decidable logic [87]

Table 4.7: $\mathcal{SHOINK}(\mathbf{D})$ concept syntax and semantics

| Constructor name | Syntax | Semantics |
|---|---|---|
| atomic concept | $A$ | $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ |
| data type | $d$ | $d^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$ |
| negation | $\neg C$ | $\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$ |
| data type negation | $\neg d$ | $\Delta_{\mathbf{D}} \backslash d^{\mathbf{D}}$ |
| conjunction | $C \sqcap D$ | $C^{\mathcal{I}} \cap D^{\mathcal{I}}$ |
| disjunction | $C \sqcup D$ | $C^{\mathcal{I}} \cup D^{\mathcal{I}}$ |
| exists restriction | $\exists R.C$ | $\{x \in \Delta^{\mathcal{I}} \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ |
| value restriction | $\forall R.C$ | $\{x \in \Delta^{\mathcal{I}} \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$ |
| exists data type restriction | $\exists R.d$ | $\{x \in \Delta^{\mathcal{I}} \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in d^{\mathbf{D}}\}$ |
| value data type restriction | $\forall R.d$ | $\{x \in \Delta^{\mathcal{I}} \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in d^{\mathbf{D}}\}$ |
| atleast restriction | $\geq nR$ | $\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}}\} \geq n\}$ |
| atmost restriction | $\leq nR$ | $\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}}\} \leq n\}$ |
| nominal | $o$ | $\sharp\{o^{\mathcal{I}}\} = 1,\ o^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ |
| concrete value | $v$ | $\sharp\{v^{\mathbf{D}}\} = 1,\ v^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$ |

*where $C$ is a concept, each $R$ is a simple role or a concrete role.*

*An interpretation $\mathcal{I}$ satisfies an expression $R_1, ..., R_n$* **Idfor** *$C$ iff*

*$\forall s, s' \in C^{\mathcal{I}}$ such that $1 \leq i \leq n$, $\langle s, t_i\rangle \in R_i^{\mathcal{I}}$ and $\langle s', t_i'\rangle \in R_i^{\mathcal{I}}$, $t_i = t_i' \forall i$ imply $s = s'$ and vice versa.*

*A* KBox *$\mathcal{K}$ is a finite set of ICs. $\mathcal{I}$ is a model of $\mathcal{K}$ iff $\mathcal{I}$ satisfies all constraints in $\mathcal{K}$. A concept $C$ is satisfiable w.r.t a KBox $\mathcal{K}$ iff $C$ and $\mathcal{K}$ have a common model. $C$ is subsumed by a concept $D$ w.r.t $\mathcal{K}$ (written $C \sqsubseteq_{\mathcal{K}} D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{K}$.*

Intuitively, an IC indicates that two instances of a concept $C$ never share the same participation in these $n$ roles, and an instance of $C$ never be identified by more than one participation in these $n$ roles. The roles must be *simple* to yield a decidable logic (as proved in Chapter 5). With this definition of $\mathcal{SHOINK}(\mathbf{D})$ semantics, we can have the mapping from the OWL-K abstract syntax of ICs to its respective DL syntax and semantics as shown in Table 4.8.

Table 4.8: Mapping from OWL-K abstract syntax to DL syntax and semantics

| Abstract syntax | DL syntax | Semantics |
|---|---|---|
| ICAssertion(ICAssertionID $C\ R_1\ ...\ R_n$) | $R_1, ..., R_n$ **Idfor** $C$ | $\mathcal{I} \models R_1, ..., R_n$ **Idfor** $C$ iff $\forall\, s, s' \in C^{\mathcal{I}}$, $1 \leq i \leq n$, $\{\langle s, t_i\rangle, \langle s', t_i'\rangle\} \in R_i^{\mathcal{I}}$, $t_i = t_i' \forall\, i$ imply $s = s'$ and vice versa |

For example, the identification constraint in Example 4.2.1 above will be represented in $\mathcal{SHOINK}(\mathbf{D})$ as

*hasFlag, onDate* **Idfor** *NationHistory*,

where *hasFlag* is an abstract role, *onDate* is a concrete role, and *NationHistory* is a concept.

With $\mathcal{SHOINK}(\mathbf{D})$-ABox defined as in Definition 3.3.5, we define $\mathcal{SHOINK}(\mathbf{D})$ knowledge bases as follows:

**Definition 4.4.5 ($\mathcal{SHOINK}(\mathbf{D})$ knowledge bases).** *A $\mathcal{SHOINK}(\mathbf{D})$ knowledge base $\Sigma$ is a tuple $\langle \mathcal{R}, \mathcal{T}, \mathcal{K}, \mathcal{A} \rangle$ where $\mathcal{R}$ is an RBox, $\mathcal{T}$ is a TBox, $\mathcal{K}$ is a KBox, and $\mathcal{A}$ is an ABox.*

*An interpretation $\mathcal{I}$ is a model of $\Sigma$ if $\mathcal{I} \models \mathcal{R}$, $\mathcal{I} \models \mathcal{T}$, $\mathcal{I} \models \mathcal{K}$ and $\mathcal{I} \models \mathcal{A}$.*

As stated in Theorem 2.3.2, $\mathcal{SHOINK}(\mathbf{D})$ KB satisfiability can be reduced to satisfiability of only RBox, TBox, and KBox.

**Definition 4.4.6.** *A TBox $\mathcal{T}$ is satisfiable w.r.t. an RBox $\mathcal{R}$ and a KBox $\mathcal{K}$ if there is a model $\mathcal{I}$ of $\mathcal{T}$, $\mathcal{R}$ and $\mathcal{K}$. A concept $C$ is* satisfiable *w.r.t. an RBox $\mathcal{R}$, a TBox $\mathcal{T}$ and a KBox $\mathcal{K}$ if there is a model $\mathcal{I}$ of $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{K}$ such that $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a model of $C$ w.r.t. $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{K}$. A concept $D$* subsumes *a concept $C$ w.r.t. $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{K}$ (written $C \sqsubseteq_{\mathcal{R},\mathcal{T},\mathcal{K}} D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds in every model $\mathcal{I}$ of $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{K}$. Two concepts $C, D$ are* equivalent *w.r.t. $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{K}$ (written $C \equiv_{\mathcal{R},\mathcal{T},\mathcal{K}} D$) if they are mutually subsuming w.r.t. $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{K}$.*

As mentioned in Chapter 3, subsumption can be reduced to satisfiability, $C \sqsubseteq_{\mathcal{R},\mathcal{T},\mathcal{K}} D$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{K}$ and $C$ is not satisfiable w.r.t $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{K}$ iff $C \sqsubseteq_{\mathcal{R},\mathcal{T},\mathcal{K}} \bot$. Moreover, concept satisfiability w.r.t $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{K}$ can be reduced to the satisfiability w.r.t $\mathcal{R}$, $\mathcal{K}$ by using the internalization technique (see section 2.3.2). Therefore, we only consider the concept satisfiability w.r.t $\mathcal{R}$ and $\mathcal{K}$.

When $\mathcal{R}$ and $\mathcal{K}$ is clear from the context, we may abuse our notation and use $\sqsubseteq$ and $\equiv$ instead of $\sqsubseteq_{\mathcal{R},\mathcal{K}}$ and $\equiv_{\mathcal{R},\mathcal{K}}$ respectively.

Note that for an IC $R$ **Idfor** $C$, a TBox can have two axioms $C \sqsubseteq \leq 1R$ and $D \sqsubseteq \geq 3R$, therefore $R$ is not required to be a functional role (cf. Table 4.7).

## 4.5 Conclusion

OWL DL is a popular resource description language in the Semantic Web environment. Therefore, to integrate relational data sources in the Semantic Web, it should be capable of expressing the notion of Identification Constraints, which have been modeled as key constraints in relational Databases. Although this standard web ontology is quite expressive, it has a very serious limitation on representing ICs.

In this chapter we have introduced a new Web ontology language OWL-K. This is an extension of OWL DL that allows one to describe various kinds of ICs. The syntax of OWL-K is both intuitive to human users and compatible with existing Web standards (such as XML, RDF(S) and OWL). Its semantics is formally specified via a new DL language $\mathcal{SHOINK}(\mathbf{D})$. This DL is an extension of $\mathcal{SHOIN}(\mathbf{D})$ with ICs. Adding ICs to DLs is nevertheless not a trivial task. Many works have tried to address this problem. Although we describe ICs in our DL in a form similar to the one in [97], our IC descriptions are more powerful in the sense that they have the capacity to fully capture all possible expressions of ICs.

The design of OWL-K shows that the language satisfies all the first six requirements presented in the end of Section 4.2. The last requirement has, however, not been handled in this chapter yet. We will deal with this one in the next chapter, showing that OWL-K's expressivity is adequate, i.e., the language is expressive enough for defining the relevant concepts in enough detail, but not too expressive to make reasoning infeasible.

CHAPTER 5

# Reasoning for OWL-K

Reasoning is important because it is necessary for the ontology design[1], ontology integration[2], as well as for ontology deployment[3]. We deal with reasoning problem for OWL-K by building a Tableau algorithm for $\mathcal{SHOIQK}(\mathbf{D})$, a language a bit more expressive than the description logic underlying OWL-K, i.e. $\mathcal{SHOINK}(\mathbf{D})$. Actually, $\mathcal{SHOINK}(\mathbf{D})$ can be considered as a $\mathcal{SHOIQK}(\mathbf{D})$ sub-language which admits the application of qualified number restrictions only to the universe ($\top$) (cf. Table 5.1). Therefore, a Tableau algorithm for $\mathcal{SHOIQK}(\mathbf{D})$ can also be applied for $\mathcal{SHOINK}(\mathbf{D})$. Moreover, while qualified number restrictions are not a part of standard OWL at the moment, they are already supported by many OWL tools (e.g. Protégé-OWL [74], Racer [59]).

Throughout this chapter, we will also consider several fragments of $\mathcal{SHOIQK}(\mathbf{D})$. By disallowing the use of data types, we obtain the DL $\mathcal{SHOIQK}$ from $\mathcal{SHOIQK}(\mathbf{D})$. The DL $\mathcal{SHOIQ}(\mathbf{D})$ is obtained from $\mathcal{SHOIQK}(\mathbf{D})$ by admitting only empty KBox. By disallowing the use of inverse roles, we obtain the fragment $\mathcal{SHOQ}(\mathbf{D})$ of $\mathcal{SHOIQ}(\mathbf{D})$. $\mathcal{SHOIQ}$ is obtained from $\mathcal{SHOIQ}(\mathbf{D})$ by

[1]e.g. checking class consistency, (unexpected) implied relationships.

[2]e.g. asserting inter-ontology relationships, computing integrated class hierarchy/consistency.

[3]e.g., determining if sets of facts are consistent w.r.t. ontology, determining if individuals are instances of ontology classes.

Table 5.1: Qualified number restrictions in $\mathcal{SHOIQK}(\mathbf{D})$

| Constructor name | Syntax | Semantics |
|---|---|---|
| atleast restriction | $\geq nR.C$ | $\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq n\}$ |
| atmost restriction | $\leq nR.C$ | $\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq n\}$ |
| atleast data type restriction | $\geq nR.d$ | $\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in d^{\mathbf{D}}\} \geq n\}$ |
| atmost data type restriction | $\leq nR.d$ | $\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in d^{\mathbf{D}}\} \leq n\}$ |

disallowing the use of data types.

We will begin with the general method of tableau algorithms, which will be presented in Section 5.1. Then in Section 5.2, we will introduce an appropriate tableau for $\mathcal{SHOIQK}(\mathbf{D})$. To construct such a tableau, first of all, we introduce some key techniques that are employed in the tableau algorithm for $\mathcal{SHOIQ}$ [84] and that influence on our construction (Section 5.3.1). To deal with the problems arising from the integration of concrete domains into $\mathcal{SHOIQ}$ (note that $\mathcal{SHOIQ}(\mathbf{D})$ has not been fully described in the literature yet) and from interactions between concrete domains and ICs, in Section 5.3.2 we introduce some assumptions about concrete domains. These assumptions are necessary for the decidability of our tableau algorithm. After that, in Section 5.3.3 we discuss the problems arising from adding ICs to $\mathcal{SHOIQ}(\mathbf{D})$ and propose a solution to build a sound and complete algorithm for reasoning problem. Section 5.3.4 is devoted to the construction of the algorithm. Section 5.3.5 introduces a reasoning procedure in the presence of concrete domains. The properties of the algorithm, namely termination, completeness and soundness, will be studied in Section 5.4, showing that in the presence of identification constraints, our algorithm still guarantees logical implication. About related works, in Section 5.5 we present the approach of Lutz et al. [97] and distinguish our algorithm from theirs. Finally, we mention the complexity of our algorithm in the conclusion.

## 5.1 General Method of Tableau Algorithms

Instead of directly testing subsumption of concepts, Tableau algorithms employ negation to reduce subsumption to concept (un)satisfiability (cf. Section 3.3.2). Tableau algorithms check the satisfiability of a concept $X$ by trying to construct a model (i.e. a witness) for $X$. The idea is using a tree to represent the model being constructed. Each node $x$ in the tree represents an individual and is labeled with a set $\mathcal{L}(x)$ of concepts of which $x$ represents an instance. Each edge $\langle x, y\rangle$ from a node $x$ to a node $y$ in the tree represents a relationship between two individuals and is labeled with a set $\mathcal{L}(\langle x, y\rangle)$ of role names of which $\langle x, y\rangle$ represents an instance[4].

If nodes $x$ and $y$ are connected by an edge $\langle x, y\rangle$, then $y$ is called a *successor* of $x$ and $x$ is called a *predecessor* of $y$; *ancestor* is the transitive closure of predecessor and *descendant* is the transitive closure of successor.

To determine the satisfiability of a concept $X$, a tree $\mathbf{T}$ is initialized to contain the root node $x$, with $\mathcal{L}(x) = \{X\}$, and expanded by repeatedly applying the *expansion rules*[5] that either extend the node labels or add new nodes. $\mathbf{T}$ is *fully expanded* or *complete* when none of the rules can be applied or an obvious contradiction, i.e *clash*, occurs. A *complete clash-free* tree $\mathbf{T}$ can be converted into a model which is a witness to the satisfiability of $X$.

To simplify the algorithm, $X$ is normalized in *negation normal form* (NNF). This means that negations ($\neg$) appear only in front of concept names.

For example, in the language $\mathcal{ALC}$ (cf. Table 3.1) we want to know whether the concept $(\exists R.A) \sqcap (\exists R.B)$ is subsumed by $\exists R.(A \sqcap B)$. This means that we must check whether the concept description

$$C := (\exists R.A) \sqcap (\exists R.B) \sqcap \neg(\exists R.(A \sqcap B))$$

is unsatisfiable. Note that in $\mathcal{ALC}$, a clash occurs when for some node $x$ on the tree $\mathbf{T}$, either $\bot \in \mathcal{L}(x)$ or $\{D, \neg D\} \in \mathcal{L}(x)$ for some concept $D$.

$C$ is first transformed into an equivalent concept $C_0$ in NNF:

$$C_0 := (\exists R.A) \sqcap (\exists R.B) \sqcap \forall R.(\neg A \sqcup \neg B))$$

The tableau algorithm for $\mathcal{ALC}$ [116] uses the expansion rules in Table 5.2 to search possible models for $C_0$ as follows:

1. Initialize a tree $\mathbf{T}$ to contain a single node $x$ labeled $\mathcal{L}(x) = \{C_0\}$
2. Apply the $\sqcap$-rule to $C_0 \in \mathcal{L}(x)$:

[4]For some simple DLs (e.g. $\mathcal{ALC}$), the set has only one element and hence each edge is labeled with a role name.

[5]Expansion rules correspond to the expressivity provided by a particular DL language.

Table 5.2: The tableau expansion rules for $\mathcal{ALC}$

| | | |
|---|---|---|
| $\sqcap$-rule: | if | $C_1 \sqcap C_2 \in \mathcal{L}(x)$, and $\{C_1, C_2\} \nsubseteq \mathcal{L}(x)$ |
| | then | $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$ |
| $\sqcup$-rule: | if | $C_1 \sqcup C_2 \in \mathcal{L}(x)$, and $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$ |
| | then | $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ where $C \in \{C_1, C_2\}$ |
| $\exists$-rule: | if | $\exists R.C \in \mathcal{L}(x)$, and there is no $y$ such that $\mathcal{L}(\langle x, y\rangle) = R$ and $C \in \mathcal{L}(y)$ |
| | then | create a new node $y$ with $\mathcal{L}(\langle x, y\rangle) = R$, $\mathcal{L}(y) = \{C\}$ |
| $\forall$-rule: | if | $\forall R.C \in \mathcal{L}(x)$, and there is some $y$ such that $\mathcal{L}(\langle x, y\rangle) = R$ and $C \notin \mathcal{L}(y)$ |
| | then | $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$ |

$$\mathcal{L}(x) = \mathcal{L}(x) \cup \{\exists R.A, \exists R.B, \forall R.(\neg A \sqcup \neg B)\},$$

3. Apply the $\exists$-rule to $\{\exists R.A, \exists R.B\} \in \mathcal{L}(x)$:

   (a) Create a new node $y$ and an edge $\langle x, y\rangle$

   $$\mathcal{L}(y) = \{A\},\ \mathcal{L}(\langle x, y\rangle) = R$$

   (b) Create a new node $z$ and an edge $\langle x, z\rangle$

   $$\mathcal{L}(z) = \{B\},\ \mathcal{L}(\langle x, z\rangle) = R$$

4. Apply the $\forall$-rule to $\forall R.(\neg A \sqcup \neg B)\} \in \mathcal{L}(x)$ with $\mathcal{L}(\langle x, y\rangle) = R$ and $\mathcal{L}(\langle x, z\rangle) = R$:

   $$\mathcal{L}(y) = \mathcal{L}(y) \cup \{\neg A \sqcup \neg B\},$$
   $$\mathcal{L}(z) = \mathcal{L}(z) \cup \{\neg A \sqcup \neg B\}$$

5. Apply the $\sqcup$-rule to $\neg A \sqcup \neg B \in \mathcal{L}(y)$:

   (a) Save **T** and try: $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\neg A\}$
   This is an obvious contradiction because $\{A, \neg A\} \in \mathcal{L}(y)$.

   (b) Restore **T** and try: $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\neg B\}$

6. Apply the $\sqcup$-rule to $\neg A \sqcup \neg B \in \mathcal{L}(z)$:

   (a) Save **T** and try: $\mathcal{L}(z) = \mathcal{L}(z) \cup \{\neg B\}$
   This is an obvious contradiction because $\{B, \neg B\} \in \mathcal{L}(z)$

   (b) Restore **T** and try: $\mathcal{L}(z) = \mathcal{L}(z) \cup \{\neg A\}$

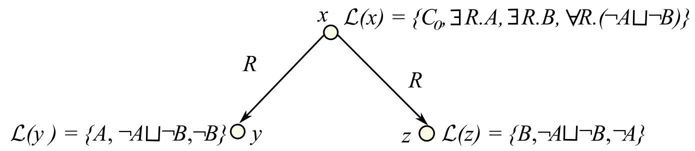

x
$\mathcal{L}(x) = \{C_0, \exists R.A, \exists R.B, \forall R.(\neg A \sqcup \neg B)\}$
R
R
$\mathcal{L}(y) = \{A, \neg A \sqcup \neg B, \neg B\}$
y
z
$\mathcal{L}(z) = \{B, \neg A \sqcup \neg B, \neg A\}$

Figure 5.1: A complete tree for $\exists R.A) \sqcap (\exists R.B) \sqcap \forall R.(\neg A \sqcup \neg B)$

None of the expansion rules are now applicable to **T**. Therefore, **T** is complet. Since no clash occurs in the complet **T**, the algorithm returns *satisfiable.* The model represented by **T** is:

$$\begin{aligned}&\Delta^{\mathcal{I}} = \{x, y, z\},\\ &A^{\mathcal{I}} = \{y\},\ B^{\mathcal{I}} = \{z\},\\ &R^{\mathcal{I}} = \{\langle x, y\rangle, \langle x, z\rangle\}.\end{aligned}$$

In the presence of transitive roles, satisfiable concepts may not have models reflected directly by trees anymore. For example, there exist some cycles in models while cycles do not exist on trees (cf. Figure 5.2). Abstractions of models, so-called *tableaux*, are then used to prove soundness and completeness of tableau algorithms. Tableau algorithms try to construct a tableau for an input concept. If the concept has a tableau, then it must have a model and vice versa.

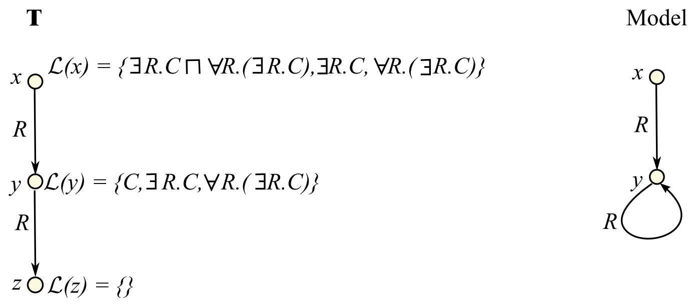

T
Model
x
$\mathcal{L}(x) = \{\exists R.C \sqcap \forall R.(\exists R.C), \exists R.C, \forall R.(\exists R.C)\}$
R
y
$\mathcal{L}(y) = \{C, \exists R.C, \forall R.(\exists R.C)\}$
R
z
$\mathcal{L}(z) = \{\}$
x
R
y
R

Figure 5.2: A complete tree and model in the form of graph for the $\mathcal{ALCH}_{R^+}$-concept $\exists R.C \sqcap \forall R.(\exists R.C)$, $R$ is a transitive role [76].

In the presence of nominals, models for satisfiable concepts can no longer be represented by trees but by graphs. A graph consists of sub-trees and an arbitrary graph connecting the roots/leaves of those trees. Note that a forest is a graph where there are no connections between leaves/roots. Procedure of building a graph is like building a tree: from an initialized system, applying expansion rules until the system is complet or a clash occurs. A complete and clash-free graph can be converted into a model which is a witness to the satisfiability of the input concept.

## 5.2 A Tableau for $\mathcal{SHOIQK}(\mathbf{D})$

We assume that all the concepts including those in KBox are in NNF. A $\mathcal{SHOIQK}(\mathbf{D})$-concept $X$ is transformed into an equivalent one in NNF by exhaustively applying the rules displayed in Table 5.3 to $X$. We use $\dot{\neg}X$ to denote the result of converting $\neg X$ into NNF.

Table 5.3: NNF rewrite rules for $\mathcal{SHOIQK}(\mathbf{D})$

| | | |
|---|---|---|
| $\neg(C \sqcap D) \equiv \neg C \sqcup \neg D$ | $\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$ | $\neg\neg C \equiv C$ |
| $\neg(\exists R.C) \equiv \forall R.\neg C$ | $\neg(\forall R.C) \equiv \exists R.\neg C$ | |
| $\neg(\exists U.d) \equiv \forall U.\neg d$ | $\neg(\forall U.d) \equiv \exists U.\neg d$ | |
| $\neg(\geq nR.C) \equiv (\leq (n-1)R.C)$ if $n \geq 1$ | $\neg(\geq 0R.C) \equiv \bot$ | |
| $\neg(\leq nR.C) \equiv (\geq (n+1)R.C)$ | | |
| $\neg(\geq nU.d) \equiv (\leq (n-1)U.d)$ if $n \geq 1$ | $\neg(\geq 0U.d) \equiv \bot$ | |
| $\neg(\leq nU.d) \equiv (\geq (n+1)U.d)$ | | |

Due to Theorem 3.3.2, $\mathcal{SHOIQK}(\mathbf{D})$-KB satisfiability can be reduced to satisfiability of TBox, RBox and KBox. In Chapter 3, we see that the internalization is used to reduce reasoning w.r.t a (possibly cyclic) Tbox to concept satisfiability. Therefore, to reduce reasoning w.r.t TBox and RBox and KBox to reasoning w.r.t Rbox and KBox only, we use an "approximation" of a universal role $\Re$ to internalize a Tbox as follows:

Let $\mathcal{R}$ be an Rbox, $\mathcal{T}$ be a Tbox, $\mathcal{K}$ be a KBox, $X$ is a concept, $o_1, ..., o_l$ are all nominals occurring in $X$, $\mathcal{T}$ or $\mathcal{K}$ and let

$$C_{\mathcal{T}} = \bigsqcap_{C_i \sqsubseteq D_i \in \mathcal{T}} \neg C_i \sqcup D_i,$$

$\Re$ be a transitive role with $R \sqsubseteq \Re$ for each $R$ occurs in $\mathcal{T}, \mathcal{R}, \mathcal{K}$, or $X$. We set

$$\mathcal{R}_{\Re} := \mathcal{R} \cup \{R \sqsubseteq \Re, R^- \sqsubseteq \Re | R \text{ occurs in } \mathcal{T}, \mathcal{R}, \mathcal{K} \text{ or } X\}.$$

$X$ is satisfiable w.r.t $\mathcal{T}$, $\mathcal{R}$, $\mathcal{K}$ iff

$$X \sqcap C_{\mathcal{T}} \sqcap \forall \mathfrak{R}.C_{\mathcal{T}} \sqcap \exists \mathfrak{R}.o_1 \sqcap ... \sqcap \exists \mathfrak{R}.o_l$$

is satisfiable w.r.t $\mathcal{R}_{\mathfrak{R}}$ and $\mathcal{K}$.

As a consequence, without loss of generality, we restrict our attention to satisfiability of $\mathcal{SHOIQK}(\mathbf{D})$-concept w.r.t an RBox and a Kbox.

When testing the satisfiability of a $\mathcal{SHOIQK}(\mathbf{D})$-concept w.r.t an RBox and a KBox, we must consider some sets of concepts defined as follows:

**Definition 5.2.1 (Set of sub-concepts).** *Let $X$ be a $\mathcal{SHOIQK}(\mathbf{D})$-concept.* sub$(X)$ *is the set of all the sub-concepts of $X$ (including $X$) and is described as follows:*

1. *if $X$ is of the form $\neg C$, $\forall R.C$, $\exists R.C$, $\leq nR.C$, $\geq nR.C$, then $C$ is a sub-concept of $X$ and* $\mathrm{sub}(X) = \{X\} \cup \mathrm{sub}(C)$,
2. *if $X$ is of the form $C \sqcap D$, $C \sqcup D$, then $C$ and $D$ are sub-concepts of $X$ and* $\mathrm{sub}(X) = \{X\} \cup \mathrm{sub}(C) \cup \mathrm{sub}(D)$,
3. *otherwise* $\mathrm{sub}(X) = \{X\}$.

Let Con$(\mathcal{K})$ be the set of concepts in a KBox $\mathcal{K}$, $\mathcal{R}$ be an RBox. We define the smallest set of "relevant" $\mathcal{SHOIQK}(\mathbf{D})$-concepts $cl(X, \mathcal{K}, \mathcal{R})$ that is closed under sub-concepts and negation as follows:

> $cl(X, \mathcal{K}, \mathcal{R}) = cl(X, \mathcal{K}) \cup \{\forall R.C | R \sqsubseteq\!\!\!\!\!\boxed{*}\; S, \forall S.C \in cl(X, \mathcal{K})\, and\, R$ in $\mathcal{R}$, $X$ or $\mathcal{K}\}$,
>
> where
>
> $cl(X, \mathcal{K}) = sub(X) \cup sub(Con(\mathcal{K})) \cup \{\dot{\neg} C | C \in \{sub(X) \cup sub(Con(\mathcal{K}))\}$ and
>
> $$sub(Con(\mathcal{K})) = \bigcup_{C \in Con(\mathcal{K})} \mathrm{sub}(C).$$

In the following, we use $cl(X)$ instead of $cl(X, \mathcal{K}, \mathcal{R})$ for short.

**Definition 5.2.2 (Tableau).** *Let $X$ be a $\mathcal{SHOIQK}(\mathbf{D})$-concept in NNF, $\mathcal{R}$ be an Rbox, $\mathcal{K}$ be a KBox, ${\mathbf{R}_A}^X, {\mathbf{R}_\mathbf{D}}^X$ be the sets of abstract and concrete roles occurring in $X$, $\mathcal{K}$ or $\mathcal{R}$, together with their inverses. A tableau $T$ for $X$ w.r.t $\mathcal{R}$ and $\mathcal{K}$ is defined as a tuple $(\mathbf{S}_A, \mathbf{S}_\mathbf{D}, \mathcal{L}, \mathcal{E}_A, \mathcal{E}_\mathbf{D})$ such that:*

- $\mathbf{S}_A$ *is a set of individuals,*
- $\mathbf{S}_{\mathbf{D}}$ *is a set of concrete values,*
- $\mathcal{L} : \mathbf{S}_A \rightarrow 2^{cl(X)}$ *maps each individual to a set of concepts which is a subset of* $cl(X)$*,*
- $\mathcal{E}_A : {\mathbf{R}_A}^X \rightarrow 2^{\mathbf{S}_A \times \mathbf{S}_A}$ *maps each abstract role to a set of pairs of individuals,*
- $\mathcal{E}_{\mathbf{D}} : {\mathbf{R}_{\mathbf{D}}}^X \rightarrow 2^{\mathbf{S}_A \times \mathbf{S}_{\mathbf{D}}}$ *maps each concrete role in* ${\mathbf{R}_{\mathbf{D}}}^X$ *to a set of pairs of an individual and a concrete value.*
- *There exists some individual* $s_0 \in \mathbf{S}_A$ *such that* $X \in \mathcal{L}(s_0)$ *and*

$\forall C, C_1, C_2 \in cl(X), R, S \in {\mathbf{R}_A}^X, U, U' \in \mathbf{R}_{\mathbf{D}}^X, s, t \in \mathbf{S}_A$ *or* $\mathbf{S}_{\mathbf{D}}$*, it holds that:*

**(P1)** *if* $C \in \mathcal{L}(s)$*, then* $\neg C \notin \mathcal{L}(s)$*,*

**(P2)** *if* $C_1 \sqcap C_2 \in \mathcal{L}(s)$*, then* $C_1 \in \mathcal{L}(s)$ *and* $C_2 \in \mathcal{L}(s)$*,*

**(P3)** *if* $C_1 \sqcup C_2 \in \mathcal{L}(s)$*, then* $C_1 \in \mathcal{L}(s)$ *or* $C_2 \in \mathcal{L}(s)$*,*

**(P4)** *if* $\langle s, t\rangle \in \mathcal{E}_A(R)$ *and* $R \sqsubseteq^{*} S$ *then* $\langle s, t\rangle \in \mathcal{E}_A(S)$*,*

**(P5)** *if* $\forall R.C \in \mathcal{L}(s)$ *and* $\langle s, t\rangle \in \mathcal{E}_A(R)$ *then* $C \in \mathcal{L}(t)$*,*

**(P6)** *if* $\exists R.C \in \mathcal{L}(s)$ *then there is some* $t \in \mathbf{S}_A$ *such that* $\langle s, t\rangle \in \mathcal{E}_A(R)$ *and* $C \in \mathcal{L}(t)$*,*

**(P7)** *if* $\forall S.C \in \mathcal{L}(s)$ *and* $\langle s, t\rangle \in \mathcal{E}_A(R)$ *for* $R \sqsubseteq^{*} S$ *with* $\mathrm{Trans}(R)$ *then* $\forall R.C \in \mathcal{L}(t)$*,*

**(P8)** $\langle s, t\rangle \in \mathcal{E}_A(R)$ *iff* $\langle t, s\rangle \in \mathcal{E}_A(\mathrm{Inv}(R))$*,*

**(P9)** *if* $\geq nS.C \in \mathcal{L}(s)$ *then* $\sharp S^T(s, C) \geq n$*,*

**(P10)** *if* $\leq nS.C \in \mathcal{L}(s)$ *then* $\sharp S^T(s, C) \leq n$*,*

**(P11)** *if* $\leq nS.C \in \mathcal{L}(s)$ *and* $\langle s, t\rangle \in \mathcal{E}_A(S)$ *then* $\{C, \dot\neg C\} \cap \mathcal{L}(t) \neq \emptyset$*,*

**(P12)** *if* $o \in \mathcal{L}(s) \cap \mathcal{L}(t)$ *then* $s = t$*,*

**(P13)** *if* $\langle s, t\rangle \in \mathcal{E}_{\mathbf{D}}(U)$ *and* $U \sqsubseteq^{*} U'$ *then* $\langle s, t\rangle \in \mathcal{E}_{\mathbf{D}}(U')$*,*

**(P14)** *if* $\forall U.d \in \mathcal{L}(s)$ *and* $\langle s, t\rangle \in \mathcal{E}_{\mathbf{D}}(U)$ *then* $t \in d^{\mathbf{D}}$*,*

**(P15)** *if* $\exists U.d \in \mathcal{L}(s)$ *then there is some* $t \in \mathbf{S}_D$ *such that* $\langle s, t\rangle \in \mathcal{E}_{\mathbf{D}}(U)$ *and* $t \in d^{\mathbf{D}}$*,*

**(P16)** *if* $\leq nU.d \in \mathcal{L}(s)$ *then* $\sharp U^T(s, d) \leq n$*,*

**(P17)** *if* $\geq nU.d \in \mathcal{L}(s)$ *then* $\sharp U^T(s, d) \geq n$*,*

**(P18)** *If* $R_1, ..., R_n$ **Idfor** $C \in \mathcal{K}$ *and* $\langle s, t_i\rangle \in \mathcal{E}_A(R_i)$ *or* $\in \mathcal{E}_{\mathbf{D}}(R_i)$ *for* $1 \leq i \leq n$ *then* $\{C, \dot\neg C\} \cap \mathcal{L}(s) \neq \emptyset$

**(P19)** *If* $R_1, ..., R_n$ **Idfor** $C \in \mathcal{K}$*,* $C \in (\mathcal{L}(s) \cap \mathcal{L}(s'))$*,* $\langle s, t_i\rangle$ *and* $\langle s', t'_i\rangle \in \mathcal{E}_A(R_i)$ *or* $\langle s, t_i\rangle$ *and* $\langle s', t'_i\rangle \in \mathcal{E}_{\mathbf{D}}(R_i)$ *and* $s = s'$ *then* $t_i = t'_i$ *for* $1 \leq i \leq n$*.*

**(P20)** *If* $R_1, ..., R_n$ **Idfor** $C \in \mathcal{K}$*,* $C \in (\mathcal{L}(s) \cap \mathcal{L}(s'))$*,* $\langle s, t_i\rangle$ *and* $\langle s', t'_i\rangle \in \mathcal{E}_A(R_i)$ *or* $\langle s, t_i\rangle$ *and* $\langle s', t'_i\rangle \in \mathcal{E}_{\mathbf{D}}(R_i)$ *and* $t_i = t'_i$ *for* $1 \leq i \leq n$ *then* $s = s'$*.*

*with*

$S^T(s, C) := \{t \in \mathbf{S}_A | \langle s, t\rangle \in \mathcal{E}_A(S) \wedge C \in \mathcal{L}(t)\}$,

$U^T(s, d) := \{t \in \mathbf{S}_\mathbf{D} | \langle s, t\rangle \in \mathcal{E}_\mathbf{D}(U) \wedge t \in d^\mathbf{D}\}$.

**Lemma 5.2.1.** *A $\mathcal{SHOIQK}(\mathbf{D})$-concept $X$ in NNF is satisfiable w.r.t an Rbox $\mathcal{R}$ and a KBox $\mathcal{K}$ iff $X$ has a tableau w.r.t $\mathcal{R}$ and $\mathcal{K}$.*

**Proof**. We concentrate on the new features w.r.t $\mathcal{SHOQ}(\mathbf{D})$ and $\mathcal{SHOIQ}$, i.e. data type number restrictions and KBox. The rest is comparable to the one found in [83, 84, 86]. Generally, the model is built from a given tableau for a concept $X$ by taking $\mathbf{S}_A$ as the interpretation domain of $X$ and adding the role-successorships for transitive roles. Then, by induction on the structure of formulas, we prove that if $C \in \mathcal{L}(s)$ then $s \in C^\mathcal{I}$ for each $C \in cl(X)$ and $s \in \mathbf{S}_A$. **(P16)** and **(P17)** ensure the correct interpretation for data type number restrictions. **(P18)**, **(P19)** and **(P20)** ensure that ICs are interpreted correctly. For the converse, each model is a tableau by definition of semantics. For completion, in the following we present the full proof.

For the "if" direction, let $T = (\mathbf{S}_A, \mathbf{S}_\mathbf{D}, \mathcal{L}, \mathcal{E}_A, \mathcal{E}_\mathbf{D})$ be a tableau for $X$ w.r.t $\mathcal{R}$ and $\mathcal{K}$. We build a model $\mathcal{I}$ for $X$ w.r.t $\mathcal{R}$ and $\mathcal{K}$:

$\Delta^\mathcal{I} := \mathbf{S}_A$,

$A^\mathcal{I} := \{s \in \mathbf{S}_A | A \in \mathcal{L}(s)\}$ for all concept name $A \in cl(X)$,

$$R^\mathcal{I} := \begin{cases} \mathcal{E}_A(R)^+ & \text{if Trans}(R) \\ \mathcal{E}_A(R) \cup \bigcup\limits_{S \underline{\ast} R, S \neq R} S^\mathcal{I} & \text{otherwise,} \end{cases}$$

$U^\mathcal{I} := \mathcal{E}_\mathbf{D}(U)$,

where $\mathcal{E}_A(R)^+$ is the transitive closure of $\mathcal{E}_A(R)$.

To show that $\mathcal{I}$ is the model of $X$ w.r.t $\mathcal{R}$ and $\mathcal{K}$, we have to prove that (i) $\mathcal{I} \models \mathcal{R}$, (ii) $X^\mathcal{I} \neq \emptyset$ and (iii) $\mathcal{I} \models \mathcal{K}$.

The definition of $\mathcal{R}^\mathcal{I}$ shows that if $\langle x, y\rangle \in R^\mathcal{I}$ then either $\langle x, y\rangle \in \mathcal{E}_A(R)^+$ in case $R$ is transitive, or $\langle x, y\rangle \in \mathcal{E}_A(R) \cup \bigcup\limits_{S \underline{\ast} R, S \neq R} S^\mathcal{I}$ to interpret correctly the non-transitive roles that can have a transitive sub-role. **(P4)** and **(P13)** of the tableau ensure that the role hierarchy is interpreted correctly. The interpretation of inverse roles is satisfied by **(P8)** of the tableau. By definition of $\mathcal{E}_A$ and $\mathcal{E}_\mathbf{D}$, $\mathcal{I} \models \mathcal{R}$.

$X^\mathcal{I} \neq \emptyset$ is proved by induction on the *norm* $\parallel X \parallel$ of $X$, which is defined as follows:

**Definition 5.2.3 (Norm of $\mathcal{SHOIQK}(\mathbf{D})$-concepts).** *Let $C, C_1, C_2$ be $\mathcal{SHOIQK}(\mathbf{D})$-concepts, $R$ be an abstract role, $U$ be a concrete role, $d$ be a data type,*

$$\begin{array}{lllll}
\parallel A \parallel & := \parallel \neg A \parallel & := 0 \text{ for a concept name } A & & \\
\parallel C_1 \sqcap C_2 \parallel & := \parallel C_1 \sqcup C_2 \parallel & := 1 + \parallel C_1 \parallel + \parallel C_2 \parallel & & \\
\parallel \leq nR.C \parallel & := \parallel \geq nR.C \parallel & := 1 + \parallel C \parallel & & \\
\parallel \exists R.C \parallel & := \parallel \forall R.C \parallel & := 1 + \parallel C \parallel & & \\
\parallel \leq nU.d \parallel & := \parallel \geq nU.d \parallel & := \parallel \forall U.d \parallel & := \parallel \exists U.d \parallel & := 1
\end{array}$$

Now we show that for each $D \in cl(X)$ and $s \in \mathbf{S}_A$, $D \in \mathcal{L}(s)$ implies $s \in D^{\mathcal{I}}$.

- If $D = A$ then by definition $s \in D^{\mathcal{I}}$.
- If $D = \neg A$ then $A \notin \mathcal{L}(s)$ by **(P1)** of Definition 5.2.2. Therefore $s \notin A^{\mathcal{I}}$.
- If $D = (C_1 \sqcap C_2)$ then due to **(P2)**, $C_1 \in \mathcal{L}(s)$ and $C_2 \in \mathcal{L}(s)$. Therefore by induction we have $s \in {C_1}^{\mathcal{I}}$ and $s \in {C_2}^{\mathcal{I}}$. Hence $s \in (C_1 \sqcap C_2)^{\mathcal{I}}$.
- If $D = (C_1 \sqcup C_2)$, $s \in (C_1 \sqcup C_2)^{\mathcal{I}}$ is proved similarly to case 3 above by **(P3)**.
- If $D = \exists R.C$ then due to **(P6)**, $D \in \mathcal{L}(s)$ implies the existence of an individual $t \in \mathbf{S}_A$ such that $\langle s, t\rangle \in \mathcal{E}_A(R)$ and $C \in \mathcal{L}(t)$. By induction we have $t \in C^{\mathcal{I}}$, by definition of $R^{\mathcal{I}}$ and **(P8)** we have $\langle s, t\rangle \in R^{\mathcal{I}}$. Hence, $s \in (\exists R.C)^{\mathcal{I}}$.
- If $D = \forall R.C$. Let $t \in \mathbf{S}_A$ be an arbitrary individual such that $\langle s, t\rangle \in R^{\mathcal{I}}$. By definition, either

  - $\langle s, t\rangle \in \mathcal{E}_A(R)$, then by **(P5)** we have $C \in \mathcal{L}(t)$, or
  - $\langle s, t\rangle \notin \mathcal{E}_A(R)$, then there exists a path $\langle s, s_1\rangle, \langle s_1, s_2\rangle, ..., \langle s_n, t\rangle \in \mathcal{E}_A(S)$ with Trans$(S)$ and $S \sqsubseteq^{*} R$. Due to **(P7)**, $\forall S.C \in \mathcal{L}(s_i)$ for $1 \leq i \leq n$. Due to **(P5)**, we have $C \in \mathcal{L}(t)$.

  In both cases, by induction we have $t \in C^{\mathcal{I}}$, hence $s \in D^{\mathcal{I}}$.

- $D = \geq nS.C$. Due to **(P9)**, we have $\sharp S^T(s, C) \geq n$. Therefore there are $n$ individuals $t_1, t_2, ..., t_n$ such that $t_i \neq t_j$ for $i \neq j$, $\langle s, t_i\rangle \in \mathcal{E}_A(S)$ and $C \in \mathcal{L}(t_i)$ for $1 \leq i \leq n$. By induction we have $t_i \in C^{\mathcal{I}}$ and because $\mathcal{E}_A(S) \subseteq S^{\mathcal{I}}$, $s \in D^{\mathcal{I}}$.

- $D =\leq nS.C$. Since $S$ is a simple role, by definition $S^{\mathcal{I}} = \mathcal{E}_A(S)$. Due to **(P10)**, we have $\sharp S^T(s,C) \leq n$. If $\sharp S^{\mathcal{I}}(s,C) > \sharp S^T(s,C)$ then there exists some $t$ with $\langle s,t\rangle \in S^{\mathcal{I}}$ and $t \in C^{\mathcal{I}}$ but $C \notin \mathcal{L}(t)$. Due to **(P11)**, $C \in \mathcal{L}(t)$ or $\dot\neg C \in \mathcal{L}(t)$ for each $t$ with $\langle s,t\rangle \in \mathcal{E}_A(S)$. Therefore, $\dot\neg C \in \mathcal{L}(t)$ that implies $t \in (\dot\neg C^{\mathcal{I}})$ by induction, contradicting $t \in C^{\mathcal{I}}$. Therefore, $\sharp S^{I}(s,C) \leq \sharp S^T(s,C) \leq n$.

- $D = o$. Let $D \in \mathcal{L}(s)$ and $D \in \mathcal{L}(t)$. Due to **(P12)**, we have $s = t$. Hence, $D$ is interpreted correctly as a nominal and $s \in D^{\mathcal{I}}$.

- $D = \exists U.d$. **(P15)** implies the existence of a value $t \in \mathbf{S_D}$ such that $\langle s,t\rangle \in \mathcal{E}_{\mathbf{D}}(U)$ and $t \in d^{\mathbf{D}}$. By definition of $U^{\mathcal{I}}$, we have $\langle s,t\rangle \in U^{\mathcal{I}}$. Hence $s \in (\exists U.d)^{\mathcal{I}}$.

- $D = \forall U.d$. Let $t \in \mathbf{S_D}$ be an arbitrary value such that $\langle s,t\rangle \in U^{\mathcal{I}}$. By definition of $U^{\mathcal{I}}$, $\langle s,t\rangle \in \mathcal{E}_{\mathbf{D}}(U)$. Therefore, due to **(P14)** $t \in d^{\mathbf{D}}$ and $s \in D^{\mathcal{I}}$.

- $D =\geq nU.d$. Due to **(P17)**, we have $\sharp U^T(s,d) \geq n$. By definition, $U^{\mathcal{I}} = \mathcal{E}_{\mathbf{D}}(U)$ so we have $\sharp\{t.\langle s,t\rangle \in U^{\mathcal{I}} \wedge t \in d^{\mathbf{D}}\} \geq n$ and $s \in D^{\mathcal{I}}$.

- $D =\leq nU.d$. Due to **(P16)**, we have $\sharp U^T(s,d) \leq n$. By definition, $U^{\mathcal{I}} = \mathcal{E}_{\mathbf{D}}(U)$, therefore $\sharp\{t.\langle s,t\rangle \in U^{\mathcal{I}} \wedge t \in d^{\mathbf{D}}\} \leq n$. Hence $s \in D^{\mathcal{I}}$.

Now it remains to demonstrate that $\mathcal{I}$ is a model of $\mathcal{K}$. Let $R_1, ..., R_n$ **Idfor** $C \in \mathcal{K}$.

Let $s, s' \in C^{\mathcal{I}}$ such that for $1 \leq i \leq n$, $\langle s,t_i\rangle$ and $\langle s',t'_i\rangle \in R_i{}^{\mathcal{I}}$, and $s = s'$. Since $R_i$ is either a simple role or a concrete role, by definition either $\langle s,t_i\rangle$ and $\langle s',t'_i\rangle \in \mathcal{E}_A(R_i)$ or $\langle s,t_i\rangle$ and $\langle s',t'_i\rangle \in \mathcal{E}_{\mathbf{D}}(R_i)$. Due to **(P18)** of Definition 5.2.2, we have $\{C, \neg C\} \cap \mathcal{L}(s) \neq \emptyset$ and $\{C, \neg C\} \cap \mathcal{L}(s') \neq \emptyset$. If $\neg C \in \mathcal{L}(s)$ then $s \in (\neg C)^{\mathcal{I}}$, in contradiction to the above assumption. Therefore we have $C \in \mathcal{L}(s)$ and $C \in \mathcal{L}(s')$ analogically. Hence $C \in (\mathcal{L}(s) \cap \mathcal{L}(s'))$. Due to **(P19)**, $t_i = t'_i$ for $1 \leq i \leq n$.

Similarly, if $s, s' \in C^{\mathcal{I}}$ such that $\langle s,t_i\rangle$ and $\langle s',t'_i\rangle \in R_i{}^{\mathcal{I}}$, and $t_i = t'_i$ for $1 \leq i \leq n$, then due to **(P20)**, we have $s = s'$.

Therefore, $\mathcal{I} \models \mathcal{K}$.

For the "only if" direction, we build a tableau for $X$ if $X$ has a model $\mathcal{I}$ w.r.t $\mathcal{R}$ and $\mathcal{K}$. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, {}^{\mathcal{I}})$ be a model of $X$ with $\mathcal{I} \models \mathcal{R}$ and $\mathcal{I} \models \mathcal{K}$. We build a tableau for $X$ $T = (\mathbf{S}_A, \mathbf{S_D}, \mathcal{L}, \mathcal{E}_A, \mathcal{E}_{\mathbf{D}})$ as follows:

$$\mathbf{S}_A := \Delta^{\mathcal{I}}$$
$$\mathbf{S}_\mathbf{D} := \Delta_\mathbf{D}$$
$$\mathcal{E}_A(R) := R^{\mathcal{I}}$$
$$\mathcal{E}_\mathbf{D}(U) := U^{\mathcal{I}}$$
$$\mathcal{L}(s) := \{C \in cl(X) | s \in C^{\mathcal{I}}\}$$

We demonstrate that $T$ is a tableau for $X$ w.r.t $\mathcal{R}$ and $\mathcal{K}$, i.e., $T$ satisfies the tableau properties.

- Except for **(P4)**, **(P13)**, **(P7)** and **(P18)-(P20)**, all properties are satisfied as a direct consequence of the definition of the interpretation $\mathcal{I}$ and of the definition of $\mathcal{L}$.
- **(P4)** and **(P13)** are satisfied because $\mathcal{I} \models \mathcal{R}$.
- For **(P7)**, let $s \in (\forall S.C)^{\mathcal{I}}$ and $\langle s, t\rangle \in R^{\mathcal{I}}$ with Trans($R$) and $R \,\underline{\circledast}\, S$. If $t \notin (\forall R.C)^{\mathcal{I}}$ then $\forall u$ such that $(t, u) \in R^{\mathcal{I}}$ (because $R$ is transitive), $u \notin C^{\mathcal{I}}$. Since $\langle s, t\rangle \in R^{\mathcal{I}}, \langle t, u\rangle \in R^{\mathcal{I}}$ and Trans($R$) so $\langle s, u\rangle \in R^{\mathcal{I}}$. We have $R \,\underline{\circledast}\, S$, so $\langle s, u\rangle \in S^{\mathcal{I}}$ and $s \notin \forall S.C$ - in contradiction to the assumption. Hence, $T$ satisfies **(P7)**.
- **(P18)**, **(P19)** and **(P20)** are satisfied by definition of the interpretation $\mathcal{I}$, of $\mathcal{E}$ and of $\mathcal{L}$ in $T$.

■

## 5.3 Constructing a $\mathcal{SHOIQK}(\mathbf{D})$-Tableau

From Lemma 5.2.1 proven above, we can check the satisfiability of a concept $X$ w.r.t an RBox and a KBox by building a tableau for $X$. If such a tableau is built successfully, the concept $X$ is satisfiable. We try to construct a tableau for a $\mathcal{SHOIQK}(\mathbf{D})$-concept by building a completion graph for it. Intuitively, in a completion graph there may exist the nodes representing individuals that are under some IC, or representing data values. Therefore it is necessary to introduce some terms:

**Definition 5.3.1 (Identifying and Identified nodes).** *Let $C$ be a concept, $R_1, ..., R_n$ be (abstract or concrete) roles with $n \geq 1$, $\mathcal{K}$ be a KBox, and* **G** *be a graph w.r.t $\mathcal{K}$. A node $x$ is called* identified *in* **G** *if there exist n nodes $y_1, ..., y_n$ in the graph such that $C \in \mathcal{L}(x)$, $R_i \in \mathcal{L}(\langle x, y_i\rangle)$ for $1 \leq i \leq n$, and $R_1, ..., R_n$* **Idfor** *$C$ in $\mathcal{K}$. Such a node $y_i$ is called an* identifying node *of $x$.*

**Definition 5.3.2 (Nodes in a graph).** *On a graph* **G***,* abstract nodes *are the nodes representing instances of concepts,* nominal nodes *represent instances of nominals,* blockable nodes *are abstract nodes that can be blocked, and* concrete nodes *are the nodes representing data values.*

Essentially, the set of abstract nodes consists of nominal and blockable nodes. The set of nominal nodes is disjoint from the set of blockable nodes. Identified nodes are abstract nodes, while identifying nodes can be either abstract or concrete nodes.

Two nodes $x$ and $y$ are called *equal* (written $x \doteq y$) if they represent the same instances of a concept (in case $x$ and $y$ are abstract nodes) or the same value (in case $x$ and $y$ are concrete nodes).

Two nodes $x$ and $y$ are called *unequal* or *distinguished* (written $x \dot{\neq} y$) if they represent distinct individuals (in case $x$ and $y$ are abstract nodes) or distinct values (in case $x$ and $y$ are concrete nodes).

### 5.3.1 Blocking and Merging

In this section, we introduce two important techniques employed in the tableau algorithm for $\mathcal{SHOIQ}$ [84], namely *pair-wise blocking* and *merging*. The former in the presence of ICs produces the problems that will be discussed in the following sections while the latter will be used to solve them.

#### 5.3.1.1 Blocking

For expressive logics (e.g. DLs including transitive roles), the expansion process for a tree (or a graph) could be repeated indefinitely. To deal with this problem, tableau algorithms employ *blocking strategy*, that is, halting the expansion process when a cycle is detected. If a node $y$ satisfies a given blocking condition then no further expansion of $y$ is performed. $y$ is then called the *blocked node.* A node $x$ is called the *blocking node* of $y$ if it blocks $y$ according to the blocking condition.

For $\mathcal{SHOIQ}$, a concept can be satisfiable but for which there exists no finite model. That is, any model of such a concept contains an infinite sequence of individuals. In order to deal with infinite models - namely to have an algorithm that terminates correctly even if the input concept has only infinite models - a *pair-wise blocking* strategy is introduced: if a path contains two pairs of successive nodes that have pairwise identical labels and whose connecting edges have identical labels, and all nodes on the path from the successor in the first pair (including itself) are blockable, then the path beyond the second pair is no longer expanded - it is blocked (cf. Figure 5.3). The successor in the second pair is called "blocked" by the successor in the first pair. The part from the blocking node to the blocked node is called a blocking cycle. A node in a finite completion graph may stand for infinitely many elements of a model. A completion graph can then be *unraveled* into an infinite model by recursively replacing the blocked node with a copy of the tree rooted at the blocking node. The identical labels make sure that copies of the blocking node and its descendants can be substituted for the blocked node and its respective descendants.

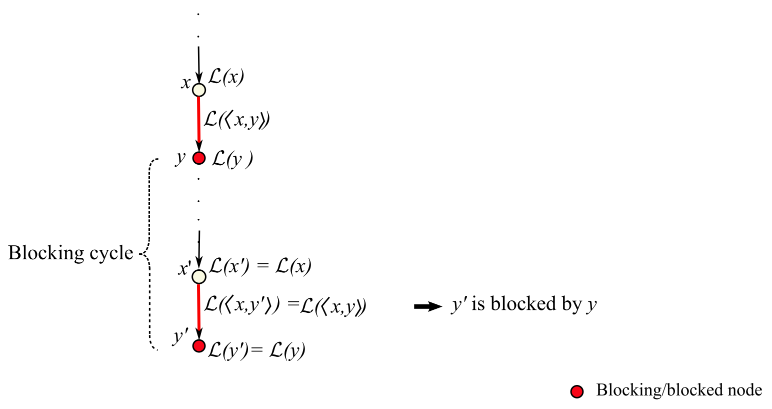


Figure 5.3: Pair-wise blocking

### 5.3.1.2 Merging

Merging defines the procedure $Merge(y, x)$ that merges $y$ into $x$ in a graph. Intuitively, when a node $y$ is merged into a node $x$, the label $\mathcal{L}(y)$ will be added to $\mathcal{L}(x)$, all edges leading to $y$ will be "moved" so that they lead to $x$, and all edges leading from $y$ to nominal nodes are "moved" so that they lead from $x$ to the same nominal nodes, then $y$ and all the blockable sub-trees below $y$ are removed from the completion graph by a procedure called $Prune(y)$. More precisely, let $V$ be a set of nodes, $E$ be a set of edges, $\mathcal{L}$ be a set of labels, and $\dot{\neq}$ be an inequality between nodes in a graph $\mathbf{G}$, $Merge(y, x)$ will merge a node $y$ into a node $x$ in $\mathbf{G}$, creating a resulting graph of $\mathbf{G}$ as shown in Table 5.4.

Table 5.4: Merging mechanism for $\mathcal{SHOIQ}$

```
Procedure Prune($y$)
  For every successor $z$ of $y$
    remove $\langle y, z\rangle$ from $E$
    If $z$ is blockable then Prune($z$)
  EndFor
  remove $y$ from $V$
End Procedure

Procedure Merge($y$,$x$)
  For every node $z$ such that $\langle z, y\rangle \in E$
    if $\{\langle x, z\rangle, \langle z, x\rangle\} \cap E = \emptyset$
      then add $\langle z, x\rangle$ to $E$ and $\mathcal{L}(\langle z, x\rangle) := \mathcal{L}(\langle z, y\rangle)$
    if $\langle z, x\rangle \in E$ then $\mathcal{L}(\langle z, x\rangle) := \mathcal{L}(\langle z, x\rangle) \cup \mathcal{L}(\langle z, y\rangle)$
    if $\langle x, z\rangle \in E$ then $\mathcal{L}(\langle x, z\rangle) := \mathcal{L}(\langle x, z\rangle) \cup \{\mathrm{Inv}(S) | S \in \mathcal{L}(\langle z, y\rangle)\}$
    remove $\langle z, y\rangle$ from $E$
  EndFor
  For every nominal node $z$ such that $\langle y, z\rangle \in E$
    if $\{\langle x, z\rangle, \langle z, x\rangle\} \cap E = \emptyset$
      then add $\langle x, z\rangle$ to $E$ and $\mathcal{L}(\langle x, z\rangle) := \mathcal{L}(\langle y, z\rangle)$
    if $\langle x, z\rangle \in E$ then $\mathcal{L}(\langle x, z\rangle) := \mathcal{L}(\langle x, z\rangle) \cup \mathcal{L}(\langle y, z\rangle)\}$
    if $\langle z, x\rangle \in E$ then $\mathcal{L}(\langle z, x\rangle) := \mathcal{L}(\langle z, x\rangle) \cup \{\mathrm{Inv}(S) | S \in \mathcal{L}(\langle y, z\rangle)\}$
    remove $\langle y, z\rangle$ from $E$
  EndFor
  $\mathcal{L}(x) := \mathcal{L}(x) \cup \mathcal{L}(y)$
  add $x \dot{\neq} t$ for every $t$ such that $y \dot{\neq} t$
  Prune($y$)
End Procedure
```

### 5.3.2 Prerequisite

We need some prerequisites before we can start constructing a $\mathcal{SHOIQK}(\mathbf{D})$-tableau. When devising a tableau algorithm for a description logic with concrete domains but without committing to a particular concrete domain, it is commonly assumed that the sets of data types over the concrete domain are conforming, which implies decidability of the satisfiability of data types conjunctions. Compared with $\mathcal{SHOQ}(\mathbf{D})$ [83] and $\mathcal{SHOIQ}$ [84], data type number restrictions (cf. Table 4.7 and 5.1) and identification constraints are new features in $\mathcal{SHOIQK}(\mathbf{D})$. This makes the assumption mentioned above is not enough for decision procedures for $\mathcal{SHOIQK}(\mathbf{D})$ as well as for $\mathcal{SHOINK}(\mathbf{D})$. In what follows, we will explain this in detail and consequently give some more assumptions.

#### 5.3.2.1 Concrete Domain and Number Restrictions

We verify the satisfiability of the concept $\geq 3U. <_2$[6] where $U$ is a concrete role and $<_2$ is a predicate name defined in the concrete domain $N = (\{<_2^{\mathbf{D}}\}, \{<_2\})$ with $<_2^{\mathbf{D}} = \{0, 1\}$.

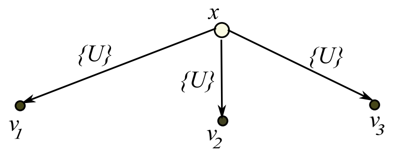


Figure 5.4: A model of the concept $\geq 3U. <_2$

Constructing a graph for this concept (cf. Figure 5.4), we see that the node $x$ must connect to at least 3 successive nodes by the edges whose labels contain $U$. Specified by number restriction, these three nodes represent three different data values. Due to the definition of universal concrete domain, this holds if there exists a solution $\delta$ that maps three variables representing these three nodes to different values. However $N$ can only provide value 0 or 1. So there are at least two nodes that share the same value. This comes into conflict with number restrictions. Therefore, to check concept satisfiability in the presence of number restrictions, a concrete domain reasoner not only find a solution $\delta$ but also should provide a DL reasoner with the value equality of variables in data type conjunctions.

**Definition 5.3.3 ($\mathcal{Q}$-conforming).** *A set of data types of a universal concrete domain* $\mathbf{D}$ *is* $\mathcal{Q}$-conforming *if there exists an algorithm that*

- *takes a finite conjunction $c$ of data types from* $\mathbf{D}$ *as input,*
- *returns clash if $c$ is unsatisfiable, otherwise*
- *returns an equality set $V_{eq} \in \mathbf{V} \times \mathbf{V}$ over the set of variables* $\mathbf{V}$ *used in $c$ such that for a solution $\delta$ for $c$ and for all $v, v' \in \mathbf{V}$, $(v, v') \in V_{eq}$ iff $\delta(v) = \delta(v')$.*

#### 5.3.2.2 Concrete Domain and ICs

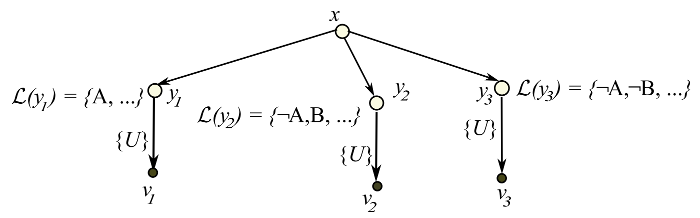
ICs in $\mathcal{SHOIQK}(\mathbf{D})$ provoke the same problem as the one shown by Lutz et al. for $\mathcal{ALCOK}(\mathcal{D})$ [97]. In particular, the problem is illustrated by checking

[6]In case of $\mathcal{SHOINK}(\mathbf{D})$, the equivalent description is $\geq 3U \sqcap \forall U. <_2$

the satisfiability of the concept (presented slightly differently from the original version):

$$\exists R.A \sqcap \exists R.(\neg A \sqcap B) \sqcap \exists R.(\neg A \sqcap \neg B) \sqcap \forall R.\exists U. <_2,$$

where $A, B$ are concepts, $R$ is an abstract role, $U$ is a concrete role, $<_2$ is the predicate name defined above.



x
$\mathcal{L}(y_1)$ = {A, ...}
$y_1$
$\mathcal{L}(y_2)$ = {¬A,B, ...}
$y_2$
$y_3$
$\mathcal{L}(y_3)$ = {¬A,¬B, ...}
{U}
{U}
{U}
$v_1$
$v_2$
$v_3$

Figure 5.5: A graph illustrating the interaction between ICs and concrete domain

The graph for this concept (cf. Figure 5.5) has three leaves whose values are 0 or 1. As a result, for every solution $\delta$ there are at least two leaves that share the same value. Those concrete nodes, as defined above, are equal. Without loss of generality, we suppose that the two equal nodes are $v_1$ and $v_2$ in the graph.

Now suppose that this concept is checked w.r.t a KBox that contains an IC $U\,\mathbf{Idfor}\,\top$. This IC requires that the predecessor of $v_1$ and of $v_2$, $y_1$ and $y_2$ respectively, must be equal. However, $y_1 \dot{\neq} y_2$ according to the construction of the graph for the concept. Consequently, the concept becomes unsatisfiable.

Similarly to the case of number restrictions above, if a concrete domain reasoner does not provide the information on variables that take the same values in its solutions, a DL reasoner may not decide on concept satisfiability. Hence, a data type reasoner should have the capability to return this information. We can use Definition 5.3.3 to define an IC conformity of a set of data types and say that a set of data types of a concrete domain $\mathbf{D}$ is $\mathcal{K}$*-conforming* if there exists an algorithm that has the behavior shown in Definition 5.3.3. Combining the condition for $\mathcal{Q}$-*conforming* and $\mathcal{K}$*-conforming*, we can say that a set of data types of a concrete domain $\mathbf{D}$ is $\mathcal{QK}$*-conforming* if there exists an algorithm that has the behavior shown in Definition 5.3.3.

We assume that the sets of data types from universal concrete domains for $\mathcal{SHOIQK}(\mathbf{D})$ are $\mathcal{QK}$-conforming.

#### 5.3.2.3 Concrete Domain, Infinite Model and ICs

Unraveling may cause an infinite number of concrete nodes because a node in a blocking cycle may have concrete nodes as its successors. Data type conjunctions, therefore, are no longer finite and the sets of data types from a universal concrete domain $\mathbf{D}$ for $\mathcal{SHOIQK}(\mathbf{D})$ are no longer ($\mathcal{QK}$-)conforming. To deal with this problem, we define an unraveled data type conjunction as follows:

**Definition 5.3.4 (Unraveled data type conjunction).** *Let* $\mathbf{D} = (\Delta_\mathbf{D}, \Phi_\mathbf{D})$ *be a universal concrete domain,* $\mathbf{V}$ *be a set of variables. For* $d \in \Phi_\mathbf{D}, v \in \mathbf{V}$*, given a sequence* $S := \{d_1(v_1), d_2(v_2), d_3(v_3)...\}$*,*

*The* $n^{th}$ *partial data type conjunction* $c_n$ *is the conjunction of the first* $n$ *terms of* $S$*:*

$$c_n = \bigwedge_{j=1}^{n} d_j(v_j), \tag{5.1}$$

*An unraveled data type conjunction* $c_\leadsto$ *is the conjunction of the terms of* $S$*.* $c_\leadsto$ *is satisfiable iff there exists a function* $\delta$ *that is a solution for* $c_i$ *for all* $i$*, i.e.,* $\delta$ *maps the variables in* $c_i$ *to data values in* $\Delta_\mathbf{D}$ *such that for* $1 \leq j \leq n$*,* $\delta(v_j) \in d_j^\mathbf{D}$*. Such a function* $\delta$ *is called a solution for* $c_\leadsto$*.*

If there is no blocking cycle, then a solution for $c_\leadsto$ is a solution for an appropriate finite data type conjunction $c$ and vice versa. When there are blocking cycles, it is obvious that a solution for $c_\leadsto$ is also a solution for $c$. Therefore, in all cases, if there is no solution for $c$ then there is no solution for $c_\leadsto$.

In the presence of ICs, the equality of concrete nodes in an unraveled "chain" may leads to unsatisfiability of the concept. To see this, consider two nodes $x$ and $x'$ in an unraveled chain (cf. Figure 5.6) such that $y_1 \doteq y'_1$, $y_2 \doteq y'_2$, and there exists $U_1, U_2\,\mathbf{Idfor}\,C$ in a KBox $\mathcal{K}$. This IC requires that $x$ and $x'$ must be equal. However, due to unraveling, $x \dot{\neq} x'$. Hence, a contradiction occurs.

Therefore, unraveling w.r.t ICs can only be satisfiable if there exists a solution that maps variables associated with infinite domains to distinct values. It is obvious that if $c_\leadsto$ is satisfiable, then such a solution is always found because we can always find a value distinct from the others in an infinite domain. In this case, the variable equality is applied only to variables associated with finite domains. Note that an infinite number of individuals cannot be identified by values in finite domains. This property is still modeled correctly with a solution mentioned above because similarly to the example in Figure 5.6, if in a blocking cycle there exists an identified node whose identifying nodes are associated with finite domains, then

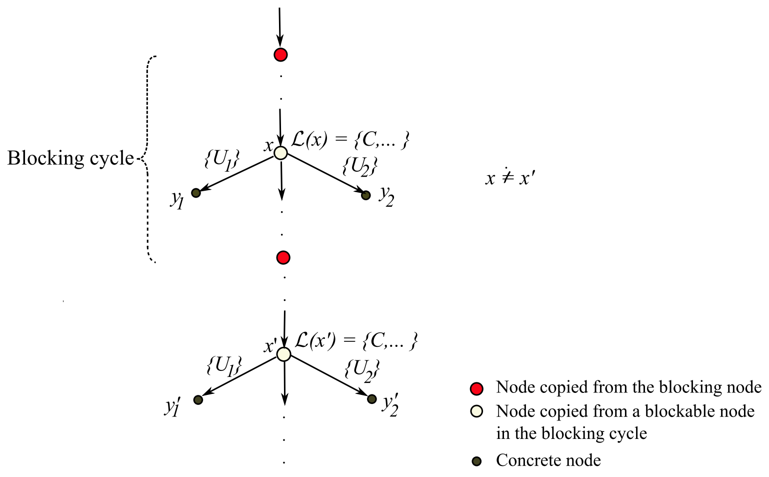


Figure 5.6: A graph illustrating the interaction between ICs and unraveling

when unraveling a contradiction will occurs.

**Definition 5.3.5 ($\mathcal{QIK}$-conforming).** *A set of data types of a universal concrete domain* $\mathbf{D}$ *is* $\mathcal{QIK}$-conforming *if there exists an algorithm that*

- *takes an unraveled data type conjunction* $c_{\leadsto}$ *from a universal concrete domain* $\mathbf{D}$ *as input,*
- *search a solution for* $c_{\leadsto}$ *that maps variables associated with infinite data type domains in* $c_{\leadsto}$ *to distinct values,*
- *returns clash if no such a solution is found, otherwise*
- *returns an equality set* $V_{eq} \in \mathbf{V} \times \mathbf{V}$ *over the set of variables* $\mathbf{V}$ *used in* $c_{\leadsto}$ *such that for a solution* $\delta$ *for* $c_{\leadsto}$ *and for all* $v, v' \in \mathbf{V}$, $(v, v') \in V_{eq}$ *iff* $\delta(v) = \delta(v')$.

Regarding to the problems shown in two previous sections, we assume that the sets of data types from universal concrete domains for $\mathcal{SHOIQK}(\mathbf{D})$ are $\mathcal{QIK}$-conforming[7].

[7]The letter $\mathcal{I}$ in "$\mathcal{QIK}$-conforming" is inspired by the interaction between inverse roles and number restrictions if unraveled chain collapses into a cycle.

### 5.3.3 ICs and Problems in Constructing a $\mathcal{SHOIQK}(\mathbf{D})$-Tableau

We extend the algorithm proposed by I. Horrocks and U. Sattler [84] for $\mathcal{SHOIQ}$ to deal with Kbox and concrete domains in $\mathcal{SHOIQK}(\mathbf{D})$. The tableau expansion rules for $\mathcal{SHOIQ}$ can be seen in Table 5.5. Before formally presenting our algorithm, we must discuss some problems that need to be overcome when trying to construct a tableau.

Table 5.5: The tableau expansion rules for $\mathcal{SHOIQ}$

| | | |
|---|---|---|
| $\sqcap$-rule: | if | $C_1 \sqcap C_2 \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and $\{C_1, C_2\} \nsubseteq \mathcal{L}(x)$ |
| | then | $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$ |
| $\sqcup$-rule: | if | $C_1 \sqcup C_2 \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$ |
| | then | $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ where $C \in \{C_1, C_2\}$ |
| $\exists$-rule: | if | $\exists R.C \in \mathcal{L}(x)$, $x$ is not blocked, and no safe $R$-neighbor $y$ of $x$ with $C \in \mathcal{L}(y)$ |
| | then | create a new node $y$ with $\mathcal{L}(\langle x, y\rangle) = \{R\}$, $\mathcal{L}(y) = \{C\}$ |
| $\forall$-rule: | if | $\forall R.C \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and there is an $R$-neighbor $y$ of $x$ with $C \notin \mathcal{L}(y)$ |
| | then | $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$ |
| $\forall_+$-rule: | if | $\forall S.C \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and there is an $R$ with Trans($R$) and $R \mathrel{\underline{\boxast}} S$, there is an $R$-neighbor $y$ of $x$ with $\forall R.C \notin \mathcal{L}(y)$ |
| | then | $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall R.C\}$ |
| *choose*-rule: | if | $\leq nR.C \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and there is an $R$-neighbor $y$ of $x$ with $\{C, \dot{\neg} C\} \cap \mathcal{L}(y) = \emptyset$ |
| | then | set $\mathcal{L}(y) = \mathcal{L} \cup \{E\}$ for some $E \in \{C, \dot{\neg} C\}$ |
| $\geq$-rule: | if | $\geq nR.C \in \mathcal{L}(x)$, $x$ is not blocked, and there are no $n$ safe $R$-neighbors $y_1, ..., y_n$ of $x$ with $C \in \mathcal{L}(y_i)$ and $y_i \dot{\neq} y_j$ $\forall 1 \leq i < j \leq n$ |
| | then | create $n$ new nodes $y_1, ..., y_n$ with $\mathcal{L}(\langle x, y_i\rangle) = \{R\}$, $\mathcal{L}(y_i) = \{C\}$, and $y_i \dot{\neq} y_j$ $\forall 1 \leq i < j \leq n$. |
| $\leq$-rule: | if | $\leq nR.C \in \mathcal{L}(z)$, $z$ is not indirectly blocked, $\sharp R^{\mathbf{G}}(z, C) > n$ and there are two $R$-neighbors $x, y$ of $z$ without $x \dot{\neq} y$, and $C \in \mathcal{L}(x) \cap \mathcal{L}(y)$ |
| | then | if $x$ is a nominal node then Merge$(y, x)$ <br> else if $y$ is a nominal node or ancestor of $x$ then Merge$(x, y)$ <br> else Merge$(y, x)$; |
| NN-rule: | if | $\leq nS.C \in \mathcal{L}(x)$, $x$ is a nominal node and there is a blockable $S$-neighbor $y$ of $x$ such that $C \in \mathcal{L}(y)$, $x$ is a successor of $y$ and, <br> there is no $m$ such that $1 \leq m \leq n$, $(\leq mS.C) \in \mathcal{L}(x)$ and <br> $m$ nominal $S$-neighbors $z_1, ..., z_m$ of $x$ with $C \in \mathcal{L}(z_i)$ and $z_i \dot{\neq} z_j$ for all $1 \leq i < j \leq m$ |
| | then | "guess" $m$ with $1 \leq m \leq n$ and set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{(\leq mS.C)\}$ <br> create $m$ new nodes $y_1, ..., y_m$ with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$, $L(y_i) = \{C, o_i\}$ $\forall o_i \in N_I$ new in $\mathbf{G}$, $y_i \dot{\neq} y_j$ for $1 \leq i < j \leq m$. |
| $o$-rule: | if | for $o \in N_I$, there are two nodes $x, y$ with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ without $x \dot{\neq} y$ |
| | then | Merge$(x, y)$ |

First of all, let us consider concrete nodes in a graph. Unlike label of an abstract node, label of a concrete node contains only data types. Therefore, merging concrete nodes does not invalidate the graph. For this reason, we merge a concrete node $y$ into a concrete node $x$ if and only if $y \doteq x$. We introduce a rule called $v$-rule to realize this merging.

Now we will see how ICs produce new problems that are actually associated with the infinite model property of $\mathcal{SHOIQK}(\mathbf{D})$. To see this, suppose that in a graph $\mathbf{G}$ we have two identified nodes $x, x'$ which represent instances of a concept $C$ such that $R_1, R_2$ **Idfor** $C$ in a Kbox, and identifying nodes $y_1, y_2, y'_1, y'_2$ such that $R_i \in \mathcal{L}(\langle x, y_i \rangle) \cap \mathcal{L}(\langle x', y'_i \rangle)$, $1 \leq i \leq 2$. If $y_1 \doteq y'_1$ and $y_2 \doteq y'_2$, then the IC requires that $x$ must be equal to $x'$. Consequently, it must be that $\mathcal{L}(x) = \mathcal{L}(x')$, $\mathcal{L}(\langle x, y_1 \rangle) = \mathcal{L}(\langle x', y'_1 \rangle)$ and $\mathcal{L}(y_1) = \mathcal{L}(y'_1)$.

If $x, y_1$ and $x'$ are ancestors of $y'_1$, $y_1$ and $y'_1$ are blockable, and all nodes on the path from $y_1$ to $y'_1$ are blockable, then due to pair-wise blocking condition $y'_1$ is blocked by $y_1$ (cf. Figure 5.7). As a consequence, $y'_1$ is considered distinguished from $y_1$ in unraveling. Hence, a contradiction occurs.

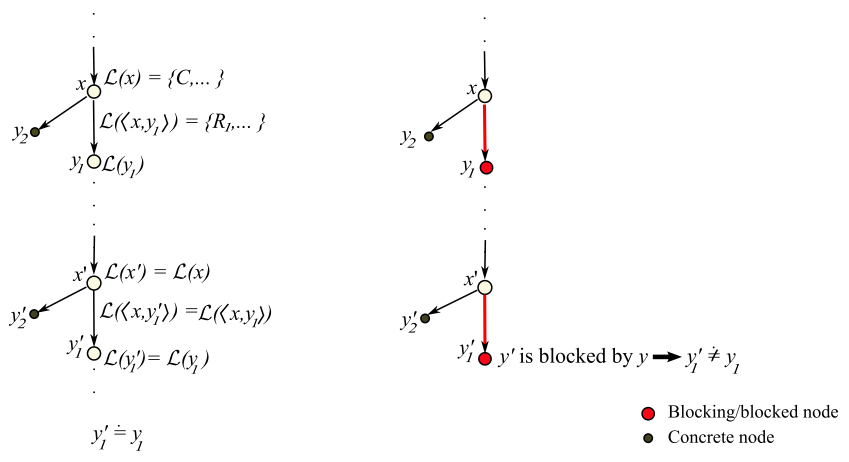


Figure 5.7: Illustration of IC satisfaction leads to blocking satisfaction

To avoid this situation, one may think of merging identified nodes before blocking condition could be checked. However, if there exist nodes $z, z'$ such that $x, x'$ are their $R$-successor respectively, and $\mathcal{L}(x)$ includes $\leq 1R^{-}.\top$, then merging may cause the individual represented by $x'$ to be related to more than one other individual by role $R$. As a consequence, the graph would be invalid (cf. Figure 5.8).

Therefore, a Tableau algorithm must be designed such that it has at the same

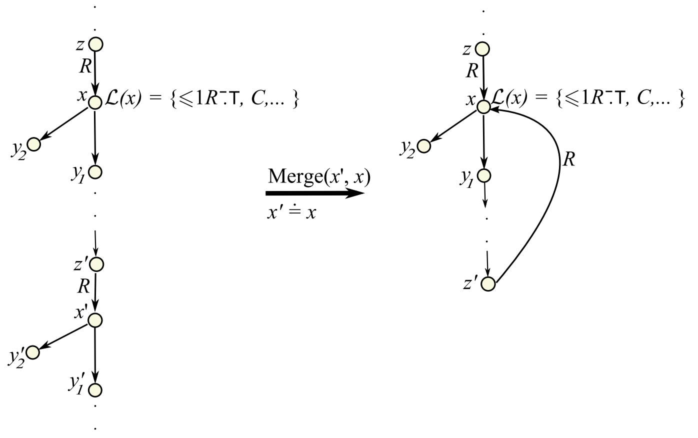

z
R
x
$\mathcal{L}(x) = \{\leqslant 1R^{-}.\top, C, \ldots\}$
$y_2$
$y_1$
z'
R
x'
$y_2'$
$y_1'$
Merge(x', x)
$x' \doteq x$
z
R
x
$\mathcal{L}(x) = \{\leqslant 1R^{-}.\top, C, \ldots\}$
$y_2$
$y_1$
R
z'

Figure 5.8: Illustration of merging identified nodes leads to an invalid graph

time the capability to handle both the satisfaction of all the constraints, including ICs, and the interaction of complex relational structures and finite tree structures (representing infinite models), while still guaranteeing termination.

To meet these requirements, we base ourselves on the decisive properties as follows:

**Lemma 5.3.1.** *In a graph constructed by a Tableau algorithm for a $\mathcal{SHOIQK}(\mathbf{D})$-concept, one node has more than one incoming edge only if it is a nominal or a concrete node.*

**Proof**. Without loss of generality, we suppose that there exists a blockable node $s$ having two incoming edges from nodes $x$ and $y$. An edge leads to an existing node if and only if merging is applied. Suppose that $s$ is the result of merging the blockable node $s'$ into $s$ (a successor of $y$ and $x$ respectively). The rules that trigger the merging include $v$-rule (introduced above), $o$- and $\leq$-rules which inherit from $\mathcal{SHOIQ}$ expansion rules (cf. Table 5.5). The $v$-rule, as explained above, only applies to concrete nodes. The $o$-rule applies only to nominal nodes. Therefore these two rules are not applied to $s$ and $s'$. By definition of the $\leq$-rule, $s$ and $s'$ must have the same parent. This comes into conflict with the assumption.

■

Due to Lemma 5.3.1, we may *nominalize* identified nodes, i.e. representing identified nodes as nominals, and merge them into one without invalidating the graph. Actually, we can make use of the rules applying to nominal nodes to handle the impact of inverse roles and number restrictions on identified nodes as shown above.

Nevertheless, there are some cases we should make clear. The first case is illustrated in Figure 5.9, where the identified nodes $x_i, x'_i$ represent instances of a concept $C$, $U$ is a concrete role, $U$ **Idfor** $C$ in a Kbox, $v_i$ are their identifying nodes, and $x_i$ is in a broken blocking cycle. The reason that a blocking cycle is broken is that nominal nodes do not exist in blocking cycles. When a node $x_i$ in a blocking cycle is nominalized, the block is no longer existent and consequently, blocking cycles are generated. The generation process, however, cannot be repeated infinitely because the sets of data types from universal concrete domains for $\mathcal{SHOIQK}(\mathbf{D})$ are $\mathcal{QIK}$-conforming. That is, two identified nodes $x_i$ and $x'_i$ share the same identified node $v_i$ only if $v_i$ is associated with a data type whose domain is finite. For any $i$ and $j$, the identifying nodes $v_i$ and $v_j$ of $x_i$ and $x_j$ (respectively) are associated also with such a data type. As a result, there must exist two nodes $v_i \doteq v_j$ and the generation of cycles terminates.

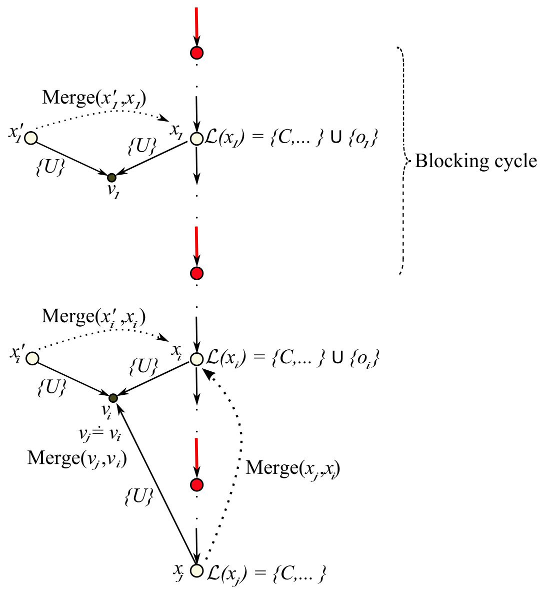


Figure 5.9: Blocking cycles are broken

The second case is illustrated in Figure 5.10, where both identified nodes $x$ and $x'$ are in a blocking cycle, $y$ is the identifying node of both $x$ and $x'$, $R$ **Idfor** $C$ in a KBox. Hence, $x$ and $x'$ must represent the same instance of $C$. We see that merging $x'$ into $x$ does not make a cycle. If $x$ and $x'$ are nominalized, the expansion of the blocking cycle will not terminate. Besides, merging these two nodes into one is not influenced by inverse roles and number restrictions, hence does not invalidate the graph. Therefore in this case we merge $x'$ into $x$ without generating a nominal.

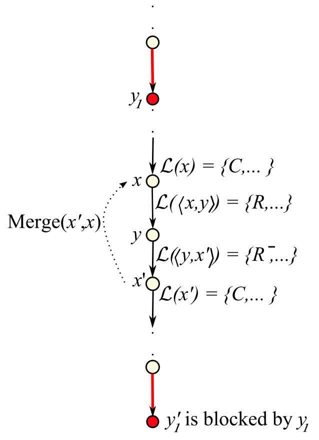

$y_1$
$\mathcal{L}(x) = \{C,\dots\}$
$x$
$\mathcal{L}(\langle x,y\rangle) = \{R,\dots\}$
Merge($x'$,$x$)
$y$
$\mathcal{L}(\langle y,x'\rangle) = \{R^-,\dots\}$
$x'$
$\mathcal{L}(x') = \{C,\dots\}$
$y_1'$ is blocked by $y_1$

Figure 5.10: Merging identified nodes without nominalization

**Lemma 5.3.2.** *Let $s$ and $s'$ be the blockable nodes satisfying an IC in a graph. Merging $s'$ and $s$ into one node does not invalidate the graph with the following merging condition:*

- *If all identifying nodes of $s$ and $s'$ are their successors and $s$ and $s'$ do not share the same predecessor then add new nominal $o_{\mathcal{K}}$ to $\mathcal{L}(s)$ and to $\mathcal{L}(s')$.*
- *otherwise merge the descendant into its ancestor.*

**Proof.** In case a new nominal is created, $s$ and $s'$ are merged by the $o$-rule and $s$ and $s'$ become the nominal node. Thus, the validation of the graph is guaranteed by the expansion rules for nominal nodes. If $s$ and $s'$ are merged without creating a nominal, then by merging condition, $s$ and $s'$ share the same predecessor or at least one of their identifying nodes $x$ is not one of their successors. If $s$ and $s'$ share the same predecessor then merging of $s$ and $s'$ does not make the blockable node $s$ have two incoming edges. In case $s$ and $s'$ do not share the same predecessor, without loss of generality we suppose that $x$ is the predecessor of $s'$. If there exists another predecessor of $s'$, by Lemma 5.3.1 $s'$ must be a nominal node. This comes into conflict with the assumption. So $s$ and $x$ must be ancestors of $s'$. This leads to the merging by the second condition which will make $s'$ and $s$ share the same predecessor. Since the blockable node $s$ has only one incoming edge (by Lemma 5.3.1), merging $s$ and $s'$ does not make a blockable node have two incoming edges.

■

The merging conditions in Lemma 5.3.2 are realized by the $\mathcal{K}$-rule. Additionally, we introduce the $\mathcal{K}_+$-rule to make sure that a node is either an identified node or not. Now we are ready to define the algorithm.

### 5.3.4 Algorithm

This section represents an algorithm to build a tableau for an input $\mathcal{SHOIQK}(\mathbf{D})$-concept.

**Definition 5.3.6 (Completion graph).** *Let $X$ be a $\mathcal{SHOIQK}(\mathbf{D})$-concept in NNF, $\mathcal{R}$ be an RBox, $\mathcal{K}$ be a KBox with its concepts in NNF.*

*A* completion graph *for $X$ w.r.t $\mathcal{R}$ and $\mathcal{K}$ is a directed graph $\mathbf{G} = (V_A, V_\mathbf{D}, E, \mathcal{L}, \dot{\neq}, \doteq)$ such that*

- *each abstract node $x \in V_A$ is labeled with a set*

$$\mathcal{L}(x) \subseteq cl(X) \cup N_I \cup \{(\leq mR.C) | (\leq nR.C) \in cl(X) \text{ and } m \leq n\} \cup \{(\leq 1R) | R \in \mathcal{K}\},$$

  *where $N_I$ is a set of nominals.*

- *each concrete node $x \in V_\mathbf{D}$ is labeled with a set $\mathcal{L}(x) \subseteq cl_\mathbf{D}(X)$ where $cl_\mathbf{D}(X)$ is the set of all the data types (possibly negative) in $cl(X)$,*
- *each edge $\langle x, y\rangle \in E$ is labeled with a set $\mathcal{L}(\langle x, y\rangle)$ of roles (possibly inverse) in $X$, $\mathcal{R}$ or $\mathcal{K}$;*
- *$\dot{\neq}$ is a binary and symmetrical relation between nodes in the graph to distinguish the unequal individuals or values,*
- *$\doteq$ is a binary and symmetrical relation between equal nodes in the graph to show their equality.*

*If $\langle x, y\rangle \in E$ then $y$ is called the* successor *of $x$ and $x$ is called the* predecessor *of $y$.* Ancestor *is the transitive closure of predecessor and* descendant *is the transitive closure of successor. A node $y$ is called an $R$-*successor *of a node $x$ if $y$ is a successor of $x$ and $S \in \mathcal{L}(\langle x, y\rangle)$ with $S \circledast R$. A node $y$ is called an $R$-*neighbor *of a node $x$ if $y$ is an $R$-successor of $x$ or if $x$ is an* $\text{Inv}(R)$*-successor of $y$.*

*If $x \in V_A$ then $x$ is either a* nominal node *or a* blockable node. *$x$ is a nominal node if $\mathcal{L}(x)$ contains a nominal. $x$ is a blockable node if it is not a nominal node.*

*A nominal $o \in N_I$ is said to be* new in $\mathbf{G}$ *if at that moment, no node in* $\mathbf{G}$ *has $o$ in its label.*

For a role $R$ and a node $x$ in $\mathbf{G}$, the set of $R$-neighbors of $x$ with $C$ in their label is defined as

$$R^{\mathbf{G}}(x, C) = \{y|y \text{ is an } R\text{-neighbor of } x \text{ and } C \in \mathcal{L}(y)\}.$$

Similarly, we define the set of $U$-successors of a node $x$ in $\mathbf{G}$, with $d$ in their label, as

$$U^{\mathbf{G}}(x, d) = \{y|y \text{ is an } U\text{-successor of } x \text{ and } d \in \mathcal{L}(y)\}.$$

**Definition 5.3.7 (Clash).** *For a node $x$, $\mathcal{L}(x)$ contains a* clash *if one of the following conditions satisfied:*

1. *For a concept name $A$, $\{A, \neg A\} \subseteq \mathcal{L}(x)$,*
2. *For an abstract role $S$ (concrete role $U$), $\leq nS.C$ ($\leq nU.d$) $\in \mathcal{L}(x)$ and there are $n+1$ $S$-neighbors ($U$-successors) $y_0, ..., y_n$ of $x$ with $C \in \mathcal{L}(y_i)$ ($d \in \mathcal{L}(y_i)$) $\forall 0 \leq i \leq n$ such that $y_i \dot{\neq} y_j$ $\forall 0 \leq i < j \leq n$,*
3. *For two concrete nodes $y$ and $x$ such that $y \dot{\neq} x$, the data type reasoner returns $y \doteq x$,*
4. *For an $o \in N_I$ there is a node $y$ such that $y \dot{\neq} x$ with $o \in \mathcal{L}(y) \cap \mathcal{L}(x)$,*
5. *$\{d_1, ..., d_n\} \in \mathcal{L}(x)$ and ${d_1}^{\mathbf{D}} \cap ... \cap {d_n}^{\mathbf{D}} = \emptyset$.*
6. *$C \in \mathcal{L}(x)$ and $R_1, ..., R_n$ **Idfor** $C \in \mathcal{K}$, there are two $R_i$-neighbors of $x$ $y_i, z_i$ for some $i$ such that $y_i \dot{\neq} z_i$.*
7. *For $x' \dot{\neq} x$ there exists a concept $C$ such that $C \in \mathcal{L}(x') \cap \mathcal{L}(x)$ and*

   *$R_1, ..., R_n$ **Idfor** $C \in \mathcal{K}$, $R_i \in \mathcal{L}(\langle x', y_i\rangle) \cap \mathcal{L}(\langle x, y_i\rangle)$ for $1 \leq i \leq n$.*
8. *For a complet graph $\mathbf{G}$, $x$ in a blocking cycle, $C \in \mathcal{L}(x)$, $U_1, ..., U_n$ are concrete roles such that $U_1, ..., U_n$ **Idfor** $C \in \mathcal{K}$, for all $i$ there are $U_i$-successor $y_i$ of $x$, $d_i \in \mathcal{L}(y_i)$ such that $d^{\mathbf{D}}{}_i$ is a finite set.*

*A graph is* clash-free *iff none of its nodes contains a clash. A graph is* complete *if no rule given in Table 5.5 and Table 5.6 is applicable or for a node $x$ in the graph, $\mathcal{L}(x)$ contains a clash.*

If $o_1, ..., o_l$ are the nominals in $X$, to check the satisfiability of the concept $X$, the algorithm initializes a completion graph

$$\mathbf{G} = (\{r_0, r_1, ..., r_l\}, \emptyset, \emptyset, \{\mathcal{L}(r_0) = \{X\}, \mathcal{L}(r_i) = \{o_i\}\}, \emptyset, \emptyset) \text{ for } 1 \leq i \leq l$$

The graph $\mathbf{G}$ is then expanded by repeatedly applying the rules given in Table 5.5 and Table 5.6 until no rule can be applied anymore or a clash occurs. If the graph is complete and clash-free then $X$ is satisfiable w.r.t $\mathcal{R}$ and $\mathcal{K}$.

Table 5.6: Additional tableau expansion rules for $\mathcal{SHOIQK}(\mathbf{D})$

| | | |
|---|---|---|
| $\exists_{\mathbf{D}}$-rule: | if | $\exists U.d \in \mathcal{L}(x)$, $x$ is not blocked , and no $U$-successor $y$ of $x$ with $d \in \mathcal{L}(y)$ |
| | then | create a new leaf $y$ with $\mathcal{L}(\langle x, y \rangle) = \{U\}$, $\mathcal{L}(y) = \{d\}$ |
| $\forall_{\mathbf{D}}$-rule: | if | $\forall U.d \in \mathcal{L}(x)$, $x$ is not indirectly blocked, and there is a $U$-successor $y$ of $x$ with $d \notin \mathcal{L}(y)$ |
| | then | $\mathcal{L}(y) = \mathcal{L}(y) \cup \{d\}$ |
| $\geq_{\mathbf{D}}$-rule: | if | $\geq nU.d \in \mathcal{L}(x)$, $x$ does not blocked, and there are no $n$ $U$-successors $y_1, ..., y_n$ of $x$ with $d \in \mathcal{L}(y_i)$ and $y_i \dot{\neq} y_j$ $\forall 1 \leq i < j \leq n$ |
| | then | create $n$ new leaves $y_1, ..., y_n$ with $\mathcal{L}(\langle x, y_i \rangle) = \{U\}$, $\mathcal{L}(y_i) = \{d\}$, and $y_i \dot{\neq} y_j$ $\forall 1 \leq i < j \leq n$. |
| $\leq_{\mathbf{D}}$-rule: | if | $\leq nU.d \in \mathcal{L}(z)$, $z$ is not indirectly blocked, $\sharp U^{\mathbf{G}}(z, d) > n$ and there are two $U$-successors $x, y$ of $z$ without $x \dot{\neq} y$, and $d \in \mathcal{L}(x) \cap \mathcal{L}(y)$ |
| | then | Merge$(y, x)$ |
| $v$-rule: | if | there are two nodes $x, y$ without $x \dot{\neq} y$, the data type reasoner returns $x \doteq y$ |
| | then | Merge$(x, y)$ |
| $\mathcal{K}_{+}$-rule: | if | $x$ has $R_i$-neighbor $y_i$ $\forall 1 \leq i \leq n$ and $R_1, ..., R_n$ **Idfor** $C \in \mathcal{K}$, $x$ is not blocked and $\{C, \dot{\neg} C\} \cap \mathcal{L}(x) = \emptyset$, |
| | then | $\mathcal{L}(x) = \mathcal{L}(x) \cup \{E\}$ for $E \in \{C, \neg C\}$ |
| $\mathcal{K}$-rule: | if | there exists $C \in \mathcal{L}(x) \cap \mathcal{L}(x')$, $R_1, ..., R_n$ **Idfor** $C \in \mathcal{K}$, and $R_i \in \mathcal{L}(\langle x, y_i \rangle) \cap \mathcal{L}(\langle x', y_i \rangle)$ $\forall 1 \leq i \leq n$ without $x \dot{\neq} x'$ |
| | then | if $x$ is a nominal node then Merge$(x', x)$ |
| | | else if $s'$ is a nominal node then Merge$(x, x')$ |
| | | else if $y_i$ is the $R_i$-successor of $x$ and of $x'$ $\forall 1 \leq i \leq n$ and predecessor of $x$ is not the predecessor of $x'$ |
| | | then $\mathcal{L}(x) = \mathcal{L}(x) \cup \{o_{\mathcal{K}}\}$ and $\mathcal{L}(x') = \mathcal{L}(x') \cup \{o_{\mathcal{K}}\}$ for $o_{\mathcal{K}} \in N_I$ new in $\mathbf{G}$ |
| | | else if $x'$ is the descendant of $x$ then Merge$(x', x)$ |
| | | else Merge$(x, x')$ |

A node is called *blocked* iff it is blocked directly or indirectly. A node $x$ is called *directly blocked* iff none of its ancestors is blocked and it has three ancestors $x', y, y'$ such that

1. $x$ is an successor of $x'$, $y$ is an successor of $y'$;
2. $y, x$ and all the nodes on the path from $y$ to $x$ are blockable;
3. $\mathcal{L}(x) = \mathcal{L}(y)$ and $\mathcal{L}(x') = \mathcal{L}(y')$;
4. $\mathcal{L}(\langle x', x\rangle) = \mathcal{L}(\langle y', y\rangle)$.

A node $x$ is *indirectly blocked* iff it is blockable and one of its ancestors is blocked. If $x$ is blocked by $y$, then we say that $y$ *blocks* $x$.

To support concrete nodes, we employ the merging mechanism presented in Table 5.4 for nodes in a graph $\mathbf{G} = (V_A, V_\mathbf{D}, E, \mathcal{L}, \dot{\neq}, \doteq)$, but modify the $Prune(y)$ procedure, where $y$ is a node in $\mathbf{G}$, as follows:

1. for every successor $z$ of $y$, remove $\langle y, z\rangle$ from $E$. If $z$ is neither a nominal nor a successor of another node then $Prune(z)$;
2. remove $y$ from $\mathbf{G}$.

If a node $y$ is merged into a node $x$, we call $x$ a *direct heir* of $y$. $y$ is called an *heir* of $x$ if it is either a direct heir of $x$ or a direct heir of a node $z$ that is an heir of $x$.

As for $\mathcal{SHOIQ}$, we use the *safe* R-neighbor to assure that enough R-neighbors of a nominal node are created. An R-neighbor $y$ of a node $x$ is called *safe* if $x$ is blockable or if $x$ is a nominal node and $y$ is not blocked.

*Level of nominal nodes.* Let $o_1, ..., o_l$ be the nominals in the input concept $X$. The *level* of a nominal node is defined inductively as follows:

- each node $x$ with $o_i \in \mathcal{L}(x), 1 \leq i \leq l$ is of level 0,
- a nominal node $x$ is of level $i$ if $x$ is not of some level $j$ with $j < i$ and $x$ has a neighbor in level $i - 1$.
- a new nominal node $x$ is of level 0 if it has no nominal neighbor.

Since identified nodes may not have nominal neighbors, we set the level 0 for them. The level of nominal nodes is used to set the priority to apply the expansion rules in the graph. It is very important for the termination of the construction. When a node in a lower level is merged into another, the level of the latter node may be reduced because merging preserves all the connections between the nominal nodes. The rules are applied to the nodes from the lower level to the higher level.

*The priority of applying the rules.* For the rules which are the same as those given for $\mathcal{SHOIQ}$, we keep the priority of application as described in [84]. The new rules for concrete domains and for ICs are applied with the lowest priority. More precisely, we have the following:

1. $o$-rule is applied with the highest priority,
2. next, applying the $\leq$- and NN-rule. They are applied first to the nominals from the lower level upwards. In case these two rules are applied to the same node, the NN-rule is applied first,
3. other rules for $\mathcal{SHOIQ}(\mathbf{D})$ are applied after that,
4. $v$-rule is applied only when two concrete nodes are equal.
5. other rules for ICs are applied with the lowest priority.

This priority strategy is crucial for the termination of the algorithm.

### 5.3.5 Concrete Domain and Reasoning Procedure

As shown above, data type number restrictions and ICs can cause the unsatisfiability of a concept. While constructing a graph, whenever new concrete nodes are created, a conflict may occur. Therefore a data type reasoner should not be executed once, after all the expansion rules have been exhaustively applied, as for $\mathcal{SHOQ}(\mathbf{D})$ but several times during the construction. Table 5.7 shows a procedure checking satisfiability for a $\mathcal{SHOIQK}(\mathbf{D})$-concept.

Table 5.7: Reasoning procedure for $\mathcal{SHOIQK}(\mathbf{D})$

```
Function Sat(G)
  If G contains a clash then return unsatisfiable
  elseif concrete nodes are generated then Check variable equality
  If G contains a clash then return unsatisfiable
  If G is complete then return satisfiable
  $G'$ := application of an extension rule to G
Return Sat($G'$)

Procedure $\mathcal{SHOIQK}(\mathbf{D})$ Reasoning procedure
  Initialize G
  Result = Sat(G)
End Procedure
```

## 5.4 Properties of the Algorithm

In order to prove that our tableau algorithm is a sound and complete decision procedure for $\mathcal{SHOIQK}(\mathbf{D})$-concept satisfiability, it is necessary to demonstrate that the models it constructs are valid w.r.t the semantics, that it will always find a model if one exists, and that it always terminates.

### 5.4.1 Termination

**Lemma 5.4.1 (Termination).** *When started with a $\mathcal{SHOINK}(\mathbf{D})$-concept $X$ in NNF, a role box $\mathcal{R}$ and a KBox $\mathcal{K}$, the Tableau algorithm terminates.*

**Proof**. Let $m = \sharp cl(X)$, $k$ be the number of abstract roles and their inverses in $X$, $\mathcal{K}$ and $\mathcal{R}$, $n_{\geq}$ be the maximum number in the atleast number restrictions, $n_{\leq}$ be the maximum number in the atmost number restrictions, $o_1, ..., o_l$ be all the nominals in $X$, $\lambda = 2^{2m+k}$ and $n_{\bowtie}$ be the maximum of $n_{\leq}$ and $n_{\geq}$.

The algorithm builds a graph that contains a set of nominal nodes interconnected arbitrarily, a set of concrete nodes, and "trees" of blockable nodes. Each "tree" is rooted at $r_0$ or at a nominal node. Each branch might end at a nominal or a concrete node.

Using the terminology proposed by Horrock et al. [84], we distinguish two categories of rules:

- *generating rules* generate the new nodes and consist of $\exists$-, $\exists_{\mathbf{D}}$-, $\geq$-, $\geq_{\mathbf{D}}$-, NN- and $\mathcal{K}$-rules;
- *shrinking rules* reduce the number of nodes and include $\leq$-, $\leq_{\mathbf{D}}$-, $o$-, $v$- and $\mathcal{K}$-rules.

Note that the $\mathcal{K}$-rule is either reduces the number of nodes (in case one of the nodes applied is nominal or the blockable nodes applied have the ancestor/descendant relationship or share the same predecessor), or increases the number of new nominal nodes by expanding the label of two blockable nodes with a new nominal. The $\mathcal{K}$-rule is not applied to the concepts in a node label but to a pair of nodes.

Termination is a consequence of the following properties of the expansion rules:

1. By rule definition, all but the shrinking rules extend the graph by creating new nodes (and edges respectively) or by extending the node label.

2. New nodes are only generated by generating rules. And these rules are applied at most once to each concept in a node label. This property is obvious when there is no merging. In case of merging, one of the three following situations occurs. Note that if a node $y$ is merged into a node $z$ and $y$ is an $S$-neighbor of $x$ then $\mathcal{L}(y)$ is added to $\mathcal{L}(z)$, $z$ inherits all the inequalities of $y$, and either $z$ is a S-neighbor of $x$ (in case $x$ is a nominal node or $y$ is a successor of $x$), or $x$ is removed by an application of $Prune(x)$ (in case $x$ is a blockable node and a successor of $y$).

   (a) For the $\exists$-rule ($\exists_{\mathbf{D}}$-rule), if $\exists S.C \in \mathcal{L}(x)$ ($\exists U.d \in \mathcal{L}(x)$) then a new node $y$ is created with $\mathcal{L}(\langle x, y\rangle) = \{S\}$ ($\mathcal{L}(\langle x, y\rangle) = \{U\}$) and $\mathcal{L}(y) = \{C\}$ ($\mathcal{L}(y) = \{d\}$). Therefore, if $y$ is merged into a node $z$, then either $x$ is removed from the graph by an application of $Prune(x)$, or $x$ has $z$ as an S-neighbor (U-successor) with $C \in \mathcal{L}(z)$ ($d \in \mathcal{L}(z)$). Consequently, the $\exists$-rule ($\exists_{\mathbf{D}}$-rule) is no longer applicable to $\exists S.C \in \mathcal{L}(x)$ ($\exists U.d \in \mathcal{L}(x)$).

   (b) For the $\geq$-rule ($\geq_{\mathbf{D}}$-rule), if $\geq nS.C \in \mathcal{L}(x)$ ($\geq nU.d \in \mathcal{L}(x)$) then $n$ new nodes $y_1, ..., y_n$ are generated with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$ ($\mathcal{L}(\langle x, y_i\rangle) = \{U\}$) , $\mathcal{L}(y_i) = \{C\}$ ($\mathcal{L}(y_i) = \{d\}$), and $y_i \dot{\neq} y_j \, \forall 1 \leq i < j \leq n$. If $y_i$ is merged into an existing node then either $x$ is removed from the graph, or $x$ has $n$ S-neighbors (U-successors) $z_1, ..., z_n$ with $C \in \mathcal{L}(z_i)$ ($d \in \mathcal{L}(z_i)$) $\forall 1 \leq i \leq n$, $z_i \dot{\neq} z_j \, \forall 1 \leq i < j \leq n$, and some $z_i$ is an heir of $y_i$ for $1 \leq i \leq n$. Consequently, the $\geq$-rule ($\geq_{\mathbf{D}}$-rule) is no longer applicable to $\geq nS.C \in \mathcal{L}(x)$ ($\geq nU.d \in \mathcal{L}(x)$).

   (c) For the NN-rule, if $\leq nS.C \in \mathcal{L}(x)$ then $m$ new nominal nodes $y_1, ..., y_m$ are created with $\mathcal{L}(\langle x, y_i\rangle) = \{S\}$, $\mathcal{L}(y_i) = \{C\}$, $y_i \dot{\neq} y_j \, \forall 1 \leq i < j \leq m$, $1 \leq m \leq n$ and $\leq mS.C \in \mathcal{L}(x)$. If $y_i$ is merged into an existing node then $\leq mS.C$ is still in $\mathcal{L}(x)$ and $x$ has $m$ nominal nodes $z_1, ..., z_m$ as S-neighbors with $C \in \mathcal{L}(z_i) \, \forall 1 \leq i \leq n$, $z_i \dot{\neq} z_j \, \forall 1 \leq i < j \leq m$, some $z_i$ is an heir of $y_i$ for $1 \leq i \leq n$. Consequently, the NN-rule is no longer applicable to $\leq nS.C \in \mathcal{L}(x)$.

   (d) As a generating rule, the $\mathcal{K}$-rule is applied at most once to a pair of blockable nodes and these nodes as well. By adding a new nominal concept to the two labels of the blockable nodes, these nodes are no longer blockable. Besides, the $o$-rule that has the highest priority has merged already these two nodes together. Consequently, the $\mathcal{K}$-rule is no longer applicable to this pair of nodes.

   As for $\mathcal{SHOIQ}$, a generating rule applied to a concept in the label of a node $x$ can create at most $n_{\geq}$ blockable successors of $x$. Since there are at most $m$ concepts in $\mathcal{L}(x)$, a node can have at most $m \times n_{\geq}$ blockable successors.

3. With the pair-wise blocking condition, the length of a path consisting entirely of blockable nodes is bounded by $\lambda$ [84].

4. The $\mathcal{K}$-rule is a special rule which is applied at most once to a pair of nodes. The reason is that in all cases, the two nodes applied are merged (either by the $\mathcal{K}$-rule itself or by the $o$-rule which is in the highest priority). This "pair", therefore, does not exist anymore. As a generating rule, the $\mathcal{K}$-rule is applied at most once to a pair of blockable nodes and to these nodes as well. The reason is that by adding a nominal to the label of these nodes, the two nodes are no longer blockable. The number of new nominal nodes created by the $\mathcal{K}$-rule is therefore bounded by a half of the number of the blockable nodes that are applied by this generating rule.

5. Let $n_{\geq U}$ be the maximum number in atleast data type number restriction. Each abstract node can have at most $m \times n_{\geq U}$ concrete successors . The number of concrete nodes is therefore bounded linearly by the number of abstract nodes.

6. Due to $\mathcal{QIK}$-conformity of data type sets, the $\mathcal{K}$-rule is not triggered for identified nodes whose identifying nodes are associated with infinite data type domains. Let $n_{\mathbf{D}}$ be the maximum number of values in the finite domains of data types in $\mathbf{D}$. A new nominal node created by the $\mathcal{K}$-rule may extend the blocking cycle at most $n_{\mathbf{D}}$ times (in case a nominal is added to a blockable node in the cycle and breaks the cycle). By property 3, this blocking cycle contains less than $\lambda$ blockable nodes (since at least one node becomes nominal). Therefore the maximum number of blockable nodes generated along a path is less than $n_{\mathbf{D}} \times \lambda$. However, for the same reason a new nominal node generated by the NN-rule may extend at most $n_{\leq}$ times the blocking cycle. So the maximum number of blockable nodes generated along this path is less than $n_{\leq} \times \lambda$. Let $N_{\leq} = Max(n_{\leq}, n_{\mathbf{D}})$. The number of blockable nodes along a path thus is bounded by $N_{\leq} \times \lambda$. By property 2, the number of paths for a "tree" is bounded by $(m \times n_{\geq})^{\lambda}$. At the beginning we have $(l+1)$ "trees". As a consequence, the number of blockable nodes in $\mathbf{G}$ is bounded by $(l+1) \times N_{\leq} \times \lambda \times (m \times n_{\geq})^{\lambda}$. A new nominal node is created by the $\mathcal{K}$-rule by adding a new nominal to the blockable predecessor of nominal or concrete nodes. Otherwise, two identified nodes cannot have the same identifying nodes (by Lemma 5.3.1). Therefore, by property 4, the number of nominal nodes generated by the $\mathcal{K}$-rule is bounded by $N_K = ((l+1)/2) \times N_{\leq} \times \lambda \times (m \times n_{\geq})^{\lambda}$.

7. The total number of nominals is bounded by $\mathcal{O}(l \times \lambda \times N_{\leq} \times (m \times n_{\bowtie})^{2\lambda})$.

Unlike for $\mathcal{SHOIQ}$, the new nominals are now generated not only by the NN-rule but also by the $\mathcal{K}$-rule. Moreover, there is an interaction between these two nominal generating sources. On the one hand, the new nominals generated by the $\mathcal{K}$-rule may be subjected to the NN-rule to create new nominals. On the other hand, the new nominals generated by the NN-rule may be the identifying nodes that make the $\mathcal{K}$-rule apply to two blockable nodes and may create a new nominal. Since the number of nominal nodes generated by the $\mathcal{K}$-rule is bounded as shown in property 6, we still have to consider the number of nominal nodes generated by the NN-rule.

Note that the NN-rule and the $\mathcal{K}$-rule are only applied after or while a nominal is added to the label of a blockable node $x$. Besides, adding a nominal to the label of a blockable node can lead to the addition of a nominal to its predecessor's label (by the $\leq$- or the $\mathcal{K}$-rule). Repeating this argument, it is possible that all the ancestors of $x$ become nominal nodes. At the beginning, there are only the nominals in level 0. Therefore, by definition of nominal node level, the application of the NN- or the $\mathcal{K}$-rule can only create nominal nodes of level 0 or 1. Since the maximum length of a path containing entirely blockable nodes is $\lambda$, the newly created nominal nodes can only be of a level below or equal to $\lambda$.

Now we calculate the upper bound of the number of nominal nodes in $\mathbf{G}$ generated by the NN-rule.

At the beginning, there are $l$ nominal nodes of level 0 in the graph. By property 6, the number of new nominal nodes of level 0 created by the $\mathcal{K}$-rule is bounded by $N_K$. The total number of nominal nodes of level 0 in $\mathbf{G}$ therefore is bounded by $N_K+l$. As a consequence, the NN-rule applies on the nominal nodes of level 0 and creates at most $(m\times n_{\leq})(N_K+l)$ nominal nodes of level 1. The $\mathcal{K}$-rule generates the nominal nodes of level 1 from blockable predecessors of nominal nodes of level 0. By property 6, the number of nominal nodes of level 1 generated by the $\mathcal{K}$-rule is bounded by $N_K$ also. Therefore, the total number of nominal nodes of level 1 in $\mathbf{G}$ is bounded by $(m \times n_{\leq})(N_K + l) + N_K$. Similarly, the nominal nodes of level $i$ are those created by the application of the NN-rule to the nodes of level $i-1$ in the graph and those of level $i$ created by the $\mathcal{K}$-rule. Consequently, the total number of nominal nodes of level $i$ in $\mathbf{G}$ is bounded by

$$l(m \times n_{\leq})^i + N_K \sum_{k=0}^{i} (m \times n_{\leq})^k.$$

The counting process terminates when the $\mathcal{K}$-rule and NN-rule have generated all the nominal nodes of level $\lambda$. The number of nominal nodes generated by the NN-rule in $\mathbf{G}$, therefore, is bounded by

$$l\sum_{i=1}^{\lambda}(m \times n_{\leq})^i + N_K \sum_{i=0}^{\lambda}\sum_{k=0}^{i}(m \times n_{\leq})^k.$$

Due to property 6, the number of nominal nodes generated by the NN-rule in $\mathbf{G}$ is bounded by $\mathcal{O}(l \times \lambda \times N_{\leq} \times (m \times n_{\bowtie})^{2\lambda})$. This is also the upper bound of the total number of nominals in $\mathbf{G}$.

■

### 5.4.2 Soundness

**Lemma 5.4.2 (Soundness).** *If the expansion rules can be applied to a concept $X$, an Rbox $\mathcal{R}$ and a KBox $\mathcal{K}$ such that they produce a complete and clash-free completion graph $\mathbf{G}$ then $X$ has a tableau w.r.t $\mathcal{R}$ and $\mathcal{K}$.*

**Proof**. The way of proving is similar to the one for $\mathcal{SHOIQ}$. Let $\mathbf{G} = (V_A, V_{\mathbf{D}}, E, \mathcal{L}, \dot{\neq}, \doteq)$. A set of *paths* of $\mathbf{G}$ is defined in order to handle the production of individuals. As sets of datatypes over the concrete domain is $\mathcal{QK}$-conforming and $\mathbf{G}$ is complet and clash-free, there exists a solution $\delta$ for the set of concrete nodes of $\mathbf{G}$. The tableau is defined from the graph $\mathbf{G}$ by taking the set of nominals and paths as the set of individuals, taking the set of values obtained by $\delta$ as the set of concrete values, and taking the label of a node as a function mapping an individual to a set of concepts. Other elements in the tableau tuple are then defined based on the given definitions. By checking the paths while unraveling the "tree" parts of $\mathbf{G}$, all the properties of the tableau are satisfied.

Precisely, a *path* is a sequence of pairs of blockable nodes of $\mathbf{G}$. The form of a path is $p := \langle(x_0, x'_0), ..., (x_n, x'_n)\rangle$. For such a path, we define $\text{Tail}(p) := x_n$ and $\text{Tail}'(p) := x'_n$. $\langle p|(x_{n+1}, x'_{n+1})\rangle$ is a path lengthened of $p$ and corresponds to $\langle(x_0, x'_0), ..., (x_n, x'_n), (x_{n+1}, x'_{n+1})\rangle$. A set of paths of $\mathbf{G}\,\text{Paths}(\mathbf{G})$ is defined inductively as follows:

- For each blockable node $x$ that is a successor of a nominal node or a root node in $\mathbf{G}$, $\langle(x, x)\rangle \in \text{Paths}(\mathbf{G})$;
- For each path $p \in \text{Paths}(\mathbf{G})$ and a blockable node $y \in \mathbf{G}$:

- if $y$ is a successor of $\text{Tail}(p)$ and $y$ is not blocked then $\langle p|(y,y)\rangle \in \text{Paths}(\mathbf{G})$, and
- if $y$ is a successor of $\text{Tail}(p)$ and $y$ is blocked by $z$, then $\langle p|(z,y)\rangle \in \text{Paths}(\mathbf{G})$

By this definition, we imply that if $p = \langle p'|(x,x')\rangle$ and $x$ is not blocked, then $x'$ is blocked iff $x' \neq x$, $x'$ is never blocked indirectly. Moreover, $\mathcal{L}(x) = \mathcal{L}(x')$ due to the blocking definition.

Let $\text{Nom}(\mathbf{G})$ be the set of nominal nodes in $\mathbf{G}$, we define a tableau for the graph $\mathbf{G}$ $T = (\mathbf{S}_A, \mathbf{S}_D, \mathcal{L}', \mathcal{E}_A, \mathcal{E}_\mathbf{D})$ as follows:

$$\begin{aligned}
&\mathbf{S}_A = \{\text{Nom}(\mathbf{G}) \cup \text{Paths}(\mathbf{G})\}\\
&\mathbf{S}_D = \{\delta(x)|x \in V_\mathbf{D}\}\\
&\mathcal{L}'(p) = \begin{cases} \mathcal{L}(\text{Tail}(p)) \text{ if } p \in \text{Paths}(\mathbf{G});\\ \mathcal{L}(p) \text{ if } p \in \text{Nom}(\mathbf{G}).\end{cases}\\
&\mathcal{E}_A(R) = \{\langle p,q\rangle \in \text{Paths}(\mathbf{G}) \times \text{Paths}(\mathbf{G})|\\
&\qquad\qquad q = \langle p|(x,x')\rangle \text{ and } x' \text{ is an R-successor of Tail}(p) \text{ or}\\
&\qquad\qquad p = \langle q|(x,x')\rangle \text{ and } x' \text{ is an Inv}(R)\text{-successor of Tail}(q)\} \cup\\
&\qquad\quad \{\langle p,x\rangle \in \text{Paths}(\mathbf{G})\times\text{Nom}(\mathbf{G})|x \text{ is an R-neighbor of Tail}(p)\} \cup\\
&\qquad\quad \{\langle x,p\rangle \in \text{Nom}(\mathbf{G})\times\text{Paths}(\mathbf{G})|\text{Tail}(p) \text{ is an R-neighbor of } x\} \cup\\
&\qquad\quad \{\langle x,y\rangle \in \text{Nom}(\mathbf{G}) \times \text{Nom}(\mathbf{G})|y \text{ is an R-neighbor of } x\}\\
&\mathcal{E}_\mathbf{D}(R) = \{\langle x,\delta(y)\rangle \ \in \mathbf{S}_A \times \mathbf{S}_\mathbf{D}|y \text{ is an R-successor of either Tail}(x) \text{ in}\\
&\text{case } x \in \text{Paths}(\mathbf{G}) \text{ or } x \text{ in case } x \in \text{Nom}(\mathbf{G})\}
\end{aligned}$$

CLAIM: $T$ is a tableau for $X$ w.r.t $\mathcal{R}$ and $\mathcal{K}$.

We demonstrate that $T$ satisfies all the properties of Definition 5.2.2.

- By definition of the algorithm, $X \in \mathcal{L}(r_0)$ or there exists an heir $x_0$ of $r_0$ such that $X \in \mathcal{L}(x_0)$. Without loss of generality, we suppose that $X \in \mathcal{L}(x_0)$. If $x_0$ is a nominal node then by definition of $\mathcal{L}'$, we have $X \in \mathcal{L}'(x_0)$. Otherwise $x_0$ is a root node that cannot be blocked. Therefore, by the definition of $\text{Paths}(\mathbf{G})$, $\langle(x_0,x_0)\rangle \in \text{Paths}(\mathbf{G})$. Consequently, by definition of Tail, we have $\text{Tail}(\langle(x_0,x_0)\rangle) = x_0$. Therefore, $X \in \mathcal{L}'(\langle(x_0,x_0)\rangle)$. As a result, in all the cases there exists an individual $s \in \mathbf{S}_A$ with $X \in \mathcal{L}'(s)$.

- **(P1)** is satisfied because $\mathbf{G}$ is clash-free. **(P2)** and **(P3)** are satisfied because Tail$(p)$ is not blocked by definition and $\mathbf{G}$ is complete.
- **(P4)** and **(P13)** are satisfied by definition of $R$-neighbor and of $U$-successor that take into account the role hierarchy, and the condition complete and clash-free of $\mathbf{G}$.
- **(P5)** Consider $\forall R.C \in \mathcal{L}'(s)$ and $\langle s, t\rangle \in \mathcal{E}_A(R)$.
  - If $\langle s, t\rangle \in \text{Paths}(\mathbf{G}) \times \text{Paths}(\mathbf{G})$ then $\forall R.C \in \mathcal{L}(\text{Tail}(s))$ by definition of $\mathcal{L}'$ and either
    - Tail$'(t)$ is an R-successor of Tail$(s)$ by definition of $\mathcal{E}_A(R)$. In this case, the complete application of rules ensures that $C \in \mathcal{L}(\text{Tail}'(t))$. By definition of path, either Tail$'(t) =$ Tail$(t)$, or $\mathcal{L}(\text{Tail}'(t)) = \mathcal{L}(\text{Tail}(t))$ because of the blocking condition. Therefore $C \in \mathcal{L}(\text{Tail}(t))$. Consequently $C \in \mathcal{L}'(t)$ by definition of $\mathcal{L}'$, or
    - Tail$'(s)$ is an Inv$(R)$-successor of Tail$(t)$. If Tail$'(s) =$ Tail$(s)$ then Tail$(s)$ is an Inv$(R)$-successor of Tail$(t)$. Hence, $C \in \mathcal{L}(\text{Tail}(t))$. If $\mathcal{L}(\text{Tail}'(s)) = \mathcal{L}(\text{Tail}(s))$ (in case the blocking condition is effected), then $\forall R.C \in \mathcal{L}(\text{Tail}'(s))$. Consequently, $C \in \mathcal{L}(\text{Tail}(t))$. By definition, $C \in \mathcal{L}'(t)$.
  - If $\langle s, t\rangle \in \text{Nom}(\mathbf{G}) \times \text{Nom}(\mathbf{G})$ then $\forall R.C \in \mathcal{L}(s)$ and $t$ is an R-neighbor of $s$. Therefore $C \in \mathcal{L}(t)$. By definition, $C \in \mathcal{L}'(t)$.
  - If $\langle s, t\rangle \in \text{Nom}(\mathbf{G}) \times \text{Paths}(\mathbf{G})$ then $\forall R.C \in \mathcal{L}(s)$. Tail$(t)$ is an R-neighbor of $s$. Therefore $C \in$ Tail$(t)$. By definition, $C \in \mathcal{L}'(t)$.
  - If $\langle s, t\rangle \in \text{Paths}(\mathbf{G}) \times \text{Nom}(\mathbf{G})$ then $\forall R.C \in \mathcal{L}(\text{Tail}(s))$. Since $t$ is an R-neighbor of Tail$(s)$, by definition of the $\forall$-rule we have $C \in \mathcal{L}(t)$. Consequently $C \in \mathcal{L}'(t)$.
- **(P6)** Consider $\exists R.C \in \mathcal{L}'(s)$ with $s \in \mathbf{S}_A$.
  - If $s \in \text{Paths}(\mathbf{G})$ then $\exists R.C \in \mathcal{L}(\text{Tail}(s))$. Since Tail$(s)$ is not blocked and the graph $\mathbf{G}$ is complete, there must exist an R-neighbor $t$ of Tail$(s)$ with $C \in \mathcal{L}(t)$.
    - if $t$ is a nominal node then $t \in S_A$ and $\mathcal{L}'(t) = \mathcal{L}(t)$. Consequently, $C \in \mathcal{L}'(t)$ and $\langle s, t\rangle \in \mathcal{E}_A(R)$.
    - if $t$ is a blockable node and an successor of Tail$(s)$ then $\langle x|(t', t)\rangle \in \mathbf{S}_A$ with either $t' = t$ or $t'$ blocks $t$, and $\langle s, \langle s|(t', t)\rangle\rangle \in \mathcal{E}_A(R)$. $\mathcal{L}'(\langle s|(t', t)\rangle) = \mathcal{L}(Tail(\langle s|(t', t)\rangle))$ so that $\mathcal{L}'(\langle s|(t', t)\rangle) = \mathcal{L}(t')$.

Consequently, $\mathcal{L}'(\langle s|(t',t)\rangle) = \mathcal{L}(t)$ because either $t' = t$ or $\mathcal{L}(t') = \mathcal{L}(t)$ by the blocking definition. As a result, we have $C \in \mathcal{L}'(\langle s|(t',t)\rangle)$.

- if $t$ is a blockable node and a predecessor of Tail($s$) then $s = \langle\langle p|(t,t)\rangle|(\text{Tail}(s), \text{Tail}'(s))\rangle$ by definition of path, where $\langle p|(t,t)\rangle$ is a path in Paths(**G**). $C \in \mathcal{L}(t)$ so that $C \in \mathcal{L}(\text{Tail}(\langle p|(t,t)\rangle))$. Consequently, $C \in \mathcal{L}'(\langle p|(t,t)\rangle)$. By definition of path, Tail($s$) is a successor of $t$. Therefore $\text{Tail}'(s) = \text{Tail}(s)$. As a result, $\langle s, \langle p|(t,t)\rangle\rangle \in \mathcal{E}_A(R)$.

– If $s \in \text{Nom}(\mathbf{G})$ then $\exists R.C \in \mathcal{L}(s)$. The graph **G** is complete. Therefore there must exist an R-successor $t$ of $s$ with $C \in \mathcal{L}(t)$.

- if $t$ is a nominal node then $\langle s,t\rangle \in \mathcal{E}_A(R)$ and $C \in \mathcal{L}'(t)$.
- if $t$ is a blockable node then $\langle s, \langle p|(t,t)\rangle\rangle \in \mathcal{E}_A(R)$ and $C \in \mathcal{L}(\text{Tail}(\langle p|(t,t)\rangle))$. Therefore $C \in \mathcal{L}'(\langle p|(t,t)\rangle)$

• **(P7)** is proved similarly to **(P5)**.

• **(P8)** is satisfied by definition of inverse role.

• **(P9)** Consider $\geq R.C \in \mathcal{L}'(s)$ with $s \in \mathbf{S}_A$.

– If $s \in \text{Nom}(\mathbf{G})$ then $\exists R.C \in \mathcal{L}(s)$. Since **G** is complete, there exists $n$ safe R-neighbors $t_1, ..., t_n$ of $s$ with $t_i \dot{\neq} t_j \, \forall 1 \leq i < j \leq n$ and $C \in \mathcal{L}(t_i) \, \forall 1 \leq i \leq n$. By construction, $t_i$ corresponds to an $y_i \in \mathbf{S}_A \, \forall 1 \leq i \leq n$ and $y_i \neq y_j \, \forall 1 \leq i \leq j \leq n$.

- if $t_i$ is a nominal node then $t_i \in \mathbf{S}_A$, $\langle s, t_i\rangle \in \mathcal{E}_A(R)$. By definition $\mathcal{L}'(t_i) = \mathcal{L}(t_i)$, hence $C \in \mathcal{L}'(t_i)$.
- if $t_i$ is a blockable node, then $t_i$ cannot be blocked because it is a safe R-neighbor of $s$. Therefore, there exists a path $\langle p|(t_i,t_i)\rangle \in$ Paths(**G**). By definition of $\mathbf{S}_A$ and $\mathcal{E}_A(R)$, we have $\langle p|(t_i,t_i)\rangle \in \mathbf{S}_A$ and $\langle s, \langle p|(t_i,t_i)\rangle\rangle \in \mathcal{E}_A(R)$. By definition, $\mathcal{L}'(\langle p|(t_i,t_i)\rangle) = \mathcal{L}(Tail(\langle p|(t_i,t_i)\rangle))$. Consequently, by definition of Tail, we have $\mathcal{L}'(\langle p|(t_i,t_i)\rangle) = \mathcal{L}(t_i)$. Therefore, $C \in \mathcal{L}'(\langle p|(t_i,t_i)\rangle)$.

– If $s \in$ Paths(**G**) then $\geq R.C \in \mathcal{L}(\text{Tail}(s))$. The graph **G** is complete. Therefore there exists $n$ R-neighbors $t_1, ..., t_n$ of Tail($s$) with $t_i \dot{\neq} t_j \, \forall 1 \leq i < j \leq n$ and $C \in \mathcal{L}(t_i) \, \forall 1 \leq i \leq n$.

- if $t_i$ is an R-successor of Tail($s$) then there exists a node $t'_i$ with $\langle s|(t'_i, t_i)\rangle \in$ Paths(**G**) and either $t'_i = t_i$ or $t_i$ is blocked by $t'_i$. Hence $\langle s|(t'_i,t_i)\rangle \in \mathbf{S}_A$ and in any case we have $\mathcal{L}(t'_i) = \mathcal{L}(t_i)$. Since

$t_i \dot{\neq} t_j\, \forall 1 \leq i < j \leq n$, we have $\langle s|(t'_i, t_i)\rangle \neq \langle s|(t'_j, t_j)\rangle \forall 1 \leq i < j \leq n$ (even if $t'_i = t'_j$). By definition, $\langle s, \langle s|(t'_i, t_i)\rangle\rangle \in \mathcal{E}_A(R)$. By definition, $\mathcal{L}'(\langle s|(t'_i, t_i)\rangle) = \mathcal{L}(\text{Tail}(\langle s|(t'_i, t_i)\rangle))$. Therefore, by definition of Tail, we have $\mathcal{L}'(\langle s|(t'_i, t_i)\rangle) = \mathcal{L}(t'_i) = \mathcal{L}(t_i)$. As a result, we have $C \in \mathcal{L}'(\langle s|(t'_i, t_i)\rangle)$.

- if Tail($s$) is an Inv(R)-successor of $t_i$, then there is at most one such $t_i$ and there exists a path $pt_i$ with $\text{Tail}(pt_i) = t_i$ and $\langle pt_i|(\text{Tail}(s), \text{Tail}(s))\rangle \in \text{Paths}(\mathbf{G})$. By definition of $\mathcal{E}_A(R)$, $\langle\langle pt_i|(\text{Tail}(s), \text{Tail}(s))\rangle, pt_i\rangle \in \mathcal{E}_A(R)$. $C \in \mathcal{L}(t_i)$ implies that $C \in \mathcal{L}(\text{Tail}(pt_i))$. By definition of $\mathcal{L}'$, $C \in \mathcal{L}'(pt_i)$. By definition, we have also $pt_i \in \mathbf{S}_A$ and $\langle s, pt_i\rangle \in \mathcal{E}_A(R)$.

- **(P10)** Consider $\leq R.C \in \mathcal{L}'(s)$ with $s \in \mathbf{S}_A$. Since Paths($\mathbf{G}$) is clash-free, there exists at most $n$ $R$-neighbors $y_i$ of $s$ with $C \in \mathcal{L}(y_i)$. By similar arguments as for **(P9)** and by construction, each $t \in \mathbf{S}_A$ with $\langle s, t\rangle \in \mathcal{E}_A(R)$ corresponds to an $R$-neighbor $y_i$ of $s$ or of Tail($s$), and none of these $R$-neighbors gives raise to more than one such $y_i$. Moreover, $\mathcal{L}'(t) = \mathcal{L}(y_i)$. Consequently, **(P10)** is satisfied.

- **(P11)** Consider $\leq R.C \in \mathcal{L}'(s)$ with $s \in \mathbf{S}_A$. **(P11)** is satisfied because Paths($\mathbf{G}$) is complete and each $t \in \mathbf{S}_A$ with $\langle s, t\rangle \in \mathcal{E}_A(R)$ corresponds to an $R$-neighbor $y_i$ of $s$ (in case $s \in \text{Nom}(\mathbf{G})$) or of Tail($s$) (in case $s \in \text{Paths}(\mathbf{G})$).

- **(P12)** is the result of the construction of the graph $\mathbf{G}$, $\mathbf{G}$ is complete and nominal nodes are not "unraveled".

- **(P14)** Consider $\forall U.d \in \mathcal{L}'(s)$ and $\langle s, v\rangle \in \mathcal{E}_\mathbf{D}(U)$ with $s \in \mathbf{S}_A$ and $v \in \mathbf{S}_\mathbf{D}$. The graph $\mathbf{G}$ is complet and clash-free. Therefore there must exist a concrete node $y$ in $\mathbf{G}$ such that $\delta(y) = v$ and $y$ is an $U$-successor of Tail($s$) or of $s$. The complet application of rules ensures that $d \in \mathcal{L}(y)$. Consequently, $v \in d^\mathbf{D}$.

- **(P15)** Consider $\exists U.d \in \mathcal{L}'(s)$ with $s \in \mathbf{S}_A$. The graph $\mathbf{G}$ is complet and clash-free. Therefore there must exist an U-successor $y$ of Tail($s$) or of $s$ with $d \in \mathcal{L}(y)$ and $\delta(y) \in \mathbf{S}_\mathbf{D}$. Consequently, by definition we have $\delta(y) \in d^\mathbf{D}$ and $\langle s, \delta(y)\rangle \in \mathcal{E}_\mathbf{D}(U)$.

- **(P16)** Consider $\leq nU.d \in \mathcal{L}'(s)$ with $s \in \mathbf{S}_A$. The graph $\mathbf{G}$ is complet and clash-free. Therefore there exists at most $n$ U-successors $y_i$ of Tail($s$) or of $s$ with $d \in \mathcal{L}(y_i)$. Consequently, by definition we have $\delta(y_i) \in \mathbf{S}_\mathbf{D}$, $\delta(y_i) \in d^\mathbf{D}$ and $\langle s, \delta(y_i)\rangle \in \mathcal{E}_\mathbf{D}(U)$ with $0 \leq i \leq n$.

- **(P17)** Consider $\geq nU.d \in \mathcal{L}'(s)$ with $s \in \mathbf{S}_A$. The graph $\mathbf{G}$ is complet and clash-free. Therefore there must exist at least $n$ U-successors $y_i$ of Tail($s$) or of $s$ with $d \in \mathcal{L}(y_i)\,\forall 1 \leq i \leq n$ and $y_i \dot{\neq} y_j\,\forall 1 \leq i < j \leq n$. Consequently, by definition we have $\delta(y_i) \in \mathbf{S_D}$, $\delta(y_i) \in d^{\mathbf{D}}$ and $\langle s, \delta(y_i)\rangle \in \mathcal{E}_{\mathbf{D}}(U)\,\forall 1 \leq i \leq n$ and $\delta(y_i) \neq \delta(y_j)\,\forall 1 \leq i < j \leq n$.

- **(P18)** is the result of the construction of the graph and of the condition complete and clash-free of $\mathbf{G}$.
- **(P19), (P20)** are satisfied by the construction of graph, by complete application of rules and by the condition complete and clash-free of $\mathbf{G}$.

■

### 5.4.3 Completeness

**Lemma 5.4.3 (Completeness).** *If $T$ is a tableau for $X$ w.r.t $\mathcal{R}$ and $\mathcal{K}$ then all the expansion rules can be applied to $X, \mathcal{R}$ and $\mathcal{K}$ such that they produce a complete and clash-free completion graph for $X$.*

**Proof**. Let $T = (\mathbf{S}_A, \mathbf{S_D}, \mathcal{L}', \mathcal{E}_A, \mathcal{E}_{\mathbf{D}})$ be a tableau for $X$ w.r.t $\mathcal{R}$ and $\mathcal{K}$. While constructing a graph, the non-deterministic rules ($\sqcup$-, *choose*-, $\leq$-, NN- and $\mathcal{K}_+$-rule) can be applied in such a way that we obtain a complete and clash-free graph. As proved in Lemma 5.4.1, the construction terminates. Therefore we obtain a completion graph.

In particular, we will use the given tableau to guide the application of the non-deterministic rules. To do this, we will inductively define a function $\pi$, mapping each abstract node in the completion graph $\mathbf{G}$ to an element of $\mathbf{S}_A$ and each concrete node in $\mathbf{G}$ to an element of $\mathbf{S_D}$, such that $\pi$ satisfies the conditions specified in Definition 5.4.1.

**Definition 5.4.1 (Conditions for $\pi$).** *For each node $x, y \in \mathbf{G}$,*

- *If $x$ is an abstract node, then $\mathcal{L}(x) \subseteq \mathcal{L}'(\pi(x))$,*
- *If $x$ is a concrete node, then $\pi(x) \in d^{\mathbf{D}}$ with $d \in \mathcal{L}(x)$,*
- *For a role $R$, if $y$ is an $R$-neighbor of $x$ then $\langle \pi(x), \pi(y)\rangle \in \mathcal{E}_A(R)$ (in case $R$ is an abstract role) or $\langle \pi(x), \pi(y)\rangle \in \mathcal{E}_{\mathbf{D}}(R)$ (in case $R$ is a concrete role),*
- *$x \dot{\neq} y$ implies $\pi(x) \neq \pi(y)$,*
- *$x \doteq y$ implies $\pi(x) = \pi(y)$.*

CLAIM: Let $\mathbf{G}$ be a completion graph and $\pi$ be a function that satisfies Definition 5.4.1. If a rule is applicable to $\mathbf{G}$ then the rule is applicable to $\mathbf{G}$ in a way that yields a completion graph $\mathbf{G}'$ and an extension of $\pi$ that satisfies Definition 5.4.1.

Let $\mathbf{G}$ be a completion graph and $\pi$ be a function that satisfies Definition 5.4.1. Consider the application of the rules:

- $\sqcap$-rule: If $C_1 \sqcap C_2 \in \mathcal{L}(x)$, then $C_1 \sqcap C_2 \in \mathcal{L}'(\pi(x))$. This implies $C_1, C_2 \in \mathcal{L}'(\pi(x))$ due to **(P2)** of Definition 5.2.2. Therefore, the rule can be applied without violating Definition 5.4.1.

- $\sqcup$-rule: If $C_1 \sqcup C_2 \in \mathcal{L}(x)$, then $C_1 \sqcup C_2 \in \mathcal{L}'(\pi(x))$. **(P3)** of Definition 5.2.2 implies $\{C_1, C_2\} \cap \mathcal{L}'(\pi(x)) \neq \emptyset$. Therefore, the $\sqcup$-rule can add a concept $C \in \{C_1, C_2\}$ to $\mathcal{L}(x)$ such that $\mathcal{L}(x) \subseteq \mathcal{L}'(\pi(x))$ holds.

- $\exists$-rule: If $\exists R.C \in \mathcal{L}(x)$, then $\exists R.C \in \mathcal{L}'(\pi(x))$. **(P6)** of Definition 5.2.2 implies that there is an element $t \in \mathbf{S}_A$ such that $\langle \pi(x), t \rangle \in \mathcal{E}_A(R)$ and $C \in \mathcal{L}'(t)$. The application of the $\exists$-rule generates a new variable $y$ with $\mathcal{L}(\langle x, y \rangle) = \{R\}$ and $\mathcal{L}(y) = \{C\}$. We set $\pi := \pi[y \mapsto t]$ which yields a function that satisfies Definition 5.4.1 for the modified graph.

- $\exists_{\mathbf{D}}$-rule: If $\exists U.d \in \mathcal{L}(x)$, then $\exists U.d \in \mathcal{L}'(\pi(x))$. **(P15)** of Definition 5.2.2 implies that there is an element $t \in \mathbf{S_D}$ such that $\langle \pi(x), t \rangle \in \mathcal{E}_{\mathbf{D}}(U)$ and $t \in d^{\mathbf{D}}$. The application of the $\exists_{\mathbf{D}}$-rule generates a new variable $y$ with $\mathcal{L}(\langle x, y \rangle) = \{U\}$ and $\mathcal{L}(y) = \{d\}$. We set $\pi := \pi[y \mapsto t]$ which yields a function that satisfies Definition 5.4.1 for the modified graph.

- $\forall$-rule: If $\forall R.C \in \mathcal{L}(x)$, then $\forall R.C \in \mathcal{L}'(\pi(x))$, and if $y$ is an $R$-neighbor of $x$, then $\langle \pi(x), \pi(y) \rangle \in \mathcal{E}_A(R)$ due to Definition 5.4.1. Consequently, by **(P5)** of Definition 5.2.2, we have $C \in \mathcal{L}'(\pi(y))$. Therefore, the $\forall$-rule can be applied without violating Definition 5.4.1.

- $\forall_{\mathbf{D}}$-rule: If $\forall U.d \in \mathcal{L}(x)$, then $\forall U.d \in \mathcal{L}'(\pi(x))$, and if $y$ is an $U$-successor of $x$, then $\langle \pi(x), \pi(y) \rangle \in \mathcal{E}_{\mathbf{D}}(R)$ due to Definition 5.4.1. Consequently, by **(P14)** of Definition 5.2.2 we have $\pi(y) \in d^{\mathbf{D}}$. Therefore, the $\forall_{\mathbf{D}}$-rule can be applied without violating Definition 5.4.1.

- $\forall_+$-rule: If $\forall S.C \in \mathcal{L}(x)$, then $\forall S.C \in \mathcal{L}'(\pi(x))$, and if there is some $R \mathrel{\underline{\boxed{\ast}}} S$ with $\text{Trans}(R)$ and $y$ is an $R$-neighbor of $x$, then $\langle \pi(x), \pi(y) \rangle \in \mathcal{E}_A(R)$ due to Definition 5.4.1. Consequently, by **(P7)** of Definition 5.2.2, we have $\forall R.C \in \mathcal{L}'(\pi(y))$. Therefore, the $\forall_+$-rule can be applied without violating Definition 5.4.1.

- *choose*-rule: If $\leq nR.C \in \mathcal{L}(x)$, then $\leq nR.C \in \mathcal{L}'(\pi(x))$, and, if there is an R-neighbor $y$ of $x$, then $\langle \pi(x), \pi(y) \rangle \in \mathcal{E}_A(R)$ due to Definition 5.4.1. Consequently, by **(P11)** of Definition 5.2.2, we have $\{C, \dot{\neg} C\} \cap \mathcal{L}'(\pi(y)) \neq \emptyset$. Therefore, the *choose*-rule can add an appropriate concept $E \in \{C, \dot{\neg} C\}$ to $\mathcal{L}(y)$ such that $\mathcal{L}(y) \subseteq \mathcal{L}'(\pi(y))$ holds.

- $\geq$-rule: If $\geq nR.C \in \mathcal{L}(x)$, then $\geq nR.C \in \mathcal{L}'(\pi(x))$. Consequently, by **(P9)** of Definition 5.2.2, we have $\sharp R^T(\pi(x), C) \geq n$. Therefore, there are $n$ individuals $t_1, ..., t_n \in \mathbf{S}_A$ such that $\langle \pi(x), t_i \rangle \in \mathcal{E}_A(R)$, $C \in \mathcal{L}'(t_i) \, \forall 1 \leq i \leq n$, and $t_i \neq t_j \, \forall 1 \leq i < j \leq n$. The $\geq$-rule generates $n$ new nodes $y_1, ..., y_n$. By setting $\pi := \pi[y_1 \mapsto t_1, ..., y_n \mapsto t_n]$, we obtain a function $\pi$ that satisfies Definition 5.4.1 for the modified graph.

- $\geq_{\mathbf{D}}$-rule: If $\geq nU.d \in \mathcal{L}(x)$, then $\geq nU.d \in \mathcal{L}'(\pi(x))$. Consequently, by **(P17)** of Definition 5.2.2, we have $\sharp U^T(\pi(x), d) \geq n$. Therefore, there are $n$ concrete values $t_1, ..., t_n \in \mathbf{S}_{\mathbf{D}}$ such that $\langle \pi(x), t_i \rangle \in \mathcal{E}_{\mathbf{D}}(R)$, $t_i \in d^{\mathbf{D}} \, \forall 1 \leq i \leq n$, and $t_i \neq t_j \, \forall 1 \leq i < j \leq n$. The $\geq_{\mathbf{D}}$-rule generates $n$ new nodes $y_1, ..., y_n$. By setting $\pi := \pi[y_1 \mapsto t_1, ..., y_n \mapsto t_n]$, we obtain a function $\pi$ that satisfies Definition 5.4.1 for the modified graph.

- $\leq$-rule: If $\leq nR.C \in \mathcal{L}(x)$, then $\leq nR.C \in \mathcal{L}'(\pi(x))$. Consequently, by **(P10)** of Definition 5.2.2, we have $\sharp R^T(\pi(x), C) \leq n$. If on the graph we have $\sharp R^{\mathbf{G}}(x, C) > n$, which implies that there are at least $n+1$ R-neighbors $y_0, ..., y_n$ of $x$ such that $C \in \mathcal{L}(y_i)$, then there must be two nodes $y, z \in \{y_0, ..., y_n\}$ such that $\pi(y) = \pi(z)$ (because otherwise $\sharp R^T(\pi(x), C) > n$ would hold). As a result, $y \dot{\neq} z$ cannot hold because of Definition 5.4.1. Therefore, the $\leq$-rule can be applied without violating Definition 5.4.1.

- $\leq_{\mathbf{D}}$-rule: If $\leq nU.d \in \mathcal{L}(x)$, then $\leq nU.d \in \mathcal{L}'(\pi(x))$. Consequently, by **(P16)** of Definition 5.2.2, we have $\sharp U^T(\pi(x), d) \leq n$. If on the graph we have $\sharp U^{\mathbf{G}}(x, d) > n$, which implies that there are at least $n+1$ U-successors $y_0, ..., y_n$ of $x$ such that $d \in \mathcal{L}(y_i)$, then there must be two nodes $y, z \in \{y_0, ..., y_n\}$ such that $\pi(y) = \pi(z)$ (because otherwise $\sharp U^T(\pi(x), d) > n$ would hold). As a result, $y \dot{\neq} z$ cannot hold because of Definition 5.4.1. Therefore, the $\leq$-rule can be applied without violating Definition 5.4.1.

- NN-rule: If $\leq nR.C \in \mathcal{L}(x)$, then $\leq nR.C \in \mathcal{L}'(\pi(x))$. Consequently, by **(P10)** of Definition 5.2.2, we have $\sharp R^T(\pi(x), C) \leq n$. The NN-rule generates $m$ new nodes $y_1, ..., y_m$ with $1 \leq m \leq n$ and $y_i \dot{\neq} y_j \, \forall 1 \leq i < j \leq m$. Suppose that there are $m$ individuals $t_1, ..., t_m \in \mathbf{S}_A$ such that $\langle \pi(x), t_i \rangle \in \mathcal{E}_A(R)$, $C \in \mathcal{L}'(t_i) \, \forall 1 \leq i \leq m$, and $t_i \neq t_j \, \forall 1 \leq i < j \leq m$. By setting $\pi := \pi[y_1 \mapsto t_1, ..., y_m \mapsto t_m]$, we obtain a function $\pi$ that satisfies Definition 5.4.1 for the modified graph.

- $o$-rule: If there are two nodes $x, y$ on the graph with a nominal concept $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$, then $o \in \mathcal{L}'(\pi(x)) \cap \mathcal{L}'(\pi(y))$. By **(P12)** of Definition 5.2.2 we have $\pi(x) = \pi(y)$. Therefore, $x \dot{\neq} y$ cannot hold because of Definition 5.4.1. Consequently, $o$-rule can be applied without violating Definition 5.4.1.

- $v$-rule: If there are two concrete nodes $x, y$ on the graph with $x \doteq y$, then $\pi(x) = \pi(y)$. Therefore, $x \dot{\neq} y$ cannot hold because of Definition 5.4.1. Consequently, $v$-rule can be applied without violating Definition 5.4.1.

- $\mathcal{K}_+$-rule: If a node $x$ has an $R_i$-neighbor $y_i \, \forall 1 \leq i \leq n$ then either $\langle \pi(x), \pi(y_i) \rangle \in \mathcal{E}_A(R_i)$ or $\langle \pi(x), \pi(y_i) \rangle \in \mathcal{E}_{\mathbf{D}}(R_i)$ by Definition 5.4.1. And if $R_1, ..., R_n$ **Idfor** $C \in \mathcal{K}$, then $\{C, \neg C\} \cap \mathcal{L}'(\pi(x)) \neq \emptyset$ by **(P18)** of Definition 5.2.2. Therefore, the $\mathcal{K}_+$-rule can add an appropriate concept $E \in \{C, \dot{\neg} C\}$ to $\mathcal{L}(x)$ such that $\mathcal{L}(x) \subseteq \mathcal{L}'(\pi(x))$ holds.

- $\mathcal{K}$-rule: If $R_i \in \mathcal{L}(\langle x, y_i \rangle) \cap \mathcal{L}(\langle x', y_i \rangle) \, \forall \, 1 \leq i \leq n$ then either $\langle \pi(x), \pi(y_i) \rangle$ and $\langle \pi(x'), \pi(y_i) \rangle \in \mathcal{E}_A(R_i)$ or $\langle \pi(x), \pi(y_i) \rangle$ and $\langle \pi(x'), \pi(y_i) \rangle \in \mathcal{E}_{\mathbf{D}}(R_i) \, \forall \, 1 \leq i \leq n$ by Definition 5.4.1. And if $C \in \mathcal{L}(x) \cap \mathcal{L}(x')$ then $C \in \mathcal{L}'(\pi(x)) \cap \mathcal{L}'(\pi(x'))$. Therefore, if $R_1, ..., R_n$ **Idfor** $C \in \mathcal{K}$ then $\pi(x) = \pi(x')$ by **(P20)** of Definition 5.2.2. Consequently, the $\mathcal{K}$-rule can be applied without violating Definition 5.4.1.

For the initial completion-graph consisting of a single node $x_0$ with $\mathcal{L}(x_0) = \{X\}$, $\dot{\neq} = \emptyset$ and $\doteq = \emptyset$, we give a function $\pi$ that satisfies Definition 5.4.1 by setting $\pi(x_0) := s_0$ for some $s_0 \in \mathbf{S}_A$ with $X \in \mathcal{L}'(s_0)$ (such an $s_0$ exists since $T$ is a tableau for $X$). Whenever a rule is applicable to $T$, it can be applied in a way that maintains the properties in Definition 5.4.1, and, since the algorithm terminates, we have that any sequence of rule applications must terminate. Definition 5.4.1 implies that any graph $\mathbf{G}$ generated by these rule applications is clash-free because it is easy to see that neither of the clashes can hold in $\mathbf{G}$:

- $\mathbf{G}$ cannot contain a node $x$ such that $\{C, \neg C\} \in \mathcal{L}(x)$ because $\mathcal{L}(x) \subseteq \mathcal{L}'(\pi(x))$. If $\{C, \neg C\} \in \mathcal{L}(x)$ holds on the graph, then **(P1)** of Definition 5.2.2 would be violated for $\pi(x)$;

- $\mathbf{G}$ cannot contain a node $x$ with $\leq nR.C$ ($\leq nU.d$) $\in \mathcal{L}(x)$ and $n+1$ $R$-neighbors ($U$-successors) $y_0, ..., y_n$ of $x$ with $C \in \mathcal{L}(y_i)$ ($d \in \mathcal{L}(y_i)$) $\forall 1 \leq i \leq n$ and $y_i \dot{\neq} y_j \, \forall \, 0 \leq i < j \leq n$ because $\leq nR.C$ ($\leq nU.d$) $\in \mathcal{L}'(\pi(x))$, and $y_i \dot{\neq} y_j$ implies $\pi(y_i) \neq \pi(y_j)$ , by Definition 5.4.1. Therefore $\sharp R^T(\pi(x), C) > n$ ($\sharp U^T(\pi(x), d) > n$), in contradiction to **(P10)** (**(P16)**) of Definition 5.2.2;

- **G** cannot contain two concrete nodes $y$ and $x$ with $y \dot{\neq} x$ and data type reasoner returns $y \doteq x$ because $y \dot{\neq} x$ implies $\pi(y) \neq \pi(x)$ by Definition 5.4.1. Data type reasoner returns $y \doteq x$ that implies $\pi(y) = \pi(x)$ by Definition 5.4.1, contradicting **(P17)** of Definition 5.2.2;

- **G** cannot contain a node $x$ satisfying the clash of form (5). Since if $d_i \in \mathcal{L}(x)$ then $\pi(x) \in d_i^{\mathbf{D}}$ for $1 \leq i \leq n$ by Definition 5.4.1. Therefore, $d_1^{\mathbf{D}} \cap ... \cap d_n^{\mathbf{D}} = \emptyset$ will contradict **(P14)**, **(P15)**, **(P17)** of Definition 5.2.2;

- **G** cannot contain a node $x$ satisfying the clash of form (6), because if $C \in \mathcal{L}(x)$ then $C \in \mathcal{L}'(\pi(x))$ by Definition 5.4.1. And if $R_1, ...R_n \,\mathbf{Idfor}\, C \in \mathcal{K}$ and there are two $R_i$-neighbors of $x$ $y_i, z_i$ for some $i$ such that $y_i \dot{\neq} z_i$, then $\langle \pi(x), \pi(y_i) \rangle, \langle \pi(x), \pi(z_i) \rangle \in \mathcal{E}_A(R)$ or $\langle \pi(x), \pi(y_i) \rangle, \langle \pi(x), \pi(z_i) \rangle \in \mathcal{E}_{\mathbf{D}}(R)$, and $\pi(y_i) \neq \pi(z_i)$ due to Definition 5.4.1, contradicting **(P19)** of Definition 5.2.2;

- **G** cannot contain a node $x$ satisfying the clash of form (7), because if $C \in \mathcal{L}(x') \cap \mathcal{L}(x)$ then $C \in \mathcal{L}'(\pi(x')) \cap \mathcal{L}'(\pi(x))$ by Definition 5.4.1. And if $R_1, ...R_n \,\mathbf{Idfor}\, C \in \mathcal{K}$ and there is an $R_i$-neighbor $y_i$ such that $R_i \in \mathcal{L}(\langle x', y_i \rangle) \cap \mathcal{L}(\langle x, y_i \rangle)$ for $1 \leq i \leq n$, then $\langle \pi(x), \pi(y_i) \rangle$ and $\langle \pi(x'), \pi(y_i) \rangle \in \mathcal{E}_A(R)$ or $\langle \pi(x), \pi(y_i) \rangle$ and $\langle \pi(x'), \pi(y_i) \rangle \in \mathcal{E}_{\mathbf{D}}(R)$ $\forall 1 \leq i \leq n$ due to Definition 5.4.1, contradicting **(P20)** of Definition 5.2.2.

- **G** cannot contain a node $x$ satisfying the clash of form (8), because if in the complet graph **G**, $x$ in a blocking cycle such that $C \in \mathcal{L}(x)$, for $1 \leq i \leq n$ $U_i$ is a concrete role, $U_1, ..., U_n \,\mathbf{Idfor}\, C \in \mathcal{K}$ and there are $n$ $U_i$-successor $y_i$ such that $U_i \in \mathcal{L}(\langle x, y_i \rangle)$, $d_i \in \mathcal{L}(y_i)$ with $d^{\mathbf{D}}{}_i$ is a finite set, then unraveling will generate a node $x' \dot{\neq} x$ such that $C \in \mathcal{L}(x')$, and $n$ $U_i$-successor $y_i'$ such that $y_i' \doteq y_i$. Due to Definition 5.4.1, $\pi(x') \neq \pi(x)$ and for $1 \leq i \leq n$ $\pi(y_i') = \pi(y_i)$, contradicting **(P20)** of Definition 5.2.2.

■

## 5.4.4 Conclusion

As an immediate consequence of Lemmas 5.2.1, 5.4.1, 5.4.2, and 5.4.3, the completion algorithm always terminates, and answers with "$X$ is satisfiable w.r.t. $\mathcal{R}$ and $\mathcal{K}$" iff $X$ is satisfiable w.r.t. $\mathcal{R}$ and $\mathcal{K}$. By Theorem 3.3.2 mentioned in Chapter 3, we can decide inference problems w.r.t. knowledge bases.

**Theorem 5.4.1.** *The Tableau algorithm presented in Section 5.3 is a decision procedure for $\mathcal{SHOIQK}(\mathbf{D})$-concept satisfiability problem w.r.t knowledge bases.*

## 5.5 Related Work

The latest effort to provide reasoning for DLs integrated with ICs was presented in [97]. The authors described two tableau-based decision procedures for concept satisfiability in $\mathcal{ALCOK}(\mathcal{D})$ w.r.t. Boolean KBox[8] and in $\mathcal{SHOQK}(\mathcal{D})$ w.r.t path-free KBox[9]. For $\mathcal{ALCOK}(\mathcal{D})$, the author employed the notion of $\mathsf{levT}(a)$ to denote the depth at which an abstract node $a$ occurs in a completion tree $\mathbf{T}$ (starting with the root node at depth 0). They defined $\prec$ as a linear ordering of abstract nodes on the tree such that $\mathsf{levT}(a) \leq \mathsf{levT}(b)$ implies $a \prec b$, and $\sim$ is an equivalence relation on the set of concrete nodes on the tree. The authors introduced also an equivalence relation $\approx_{\mathsf{a}}$ on abstract nodes, which is induced by the relation $\sim$, and yields the equivalence relation on concrete nodes $\approx_c \supseteq \sim$. For each role name $R$, an abstract node $b$ is an $R/\approx_{\mathsf{a}}$-neighbor of an abstract node $a$ if there exists an abstract node $c$ such that $a \approx_{\mathsf{a}} c$ and $b$ is an $R$-successor of $c$. $x \approx_c y$ if $x \sim y$ or there are an abstract node $a$ and a concrete role $g$ such that $x$ and $y$ are $g/=\approx_{\mathsf{a}}$-neighbors of $a$. Key assertions (i.e. ICs in our terminology) are used to define the $\approx_{\mathsf{a}}$ relation.

When an abstract node $b$ is added to a tree, it is inserted into $\prec$ such that $b \prec c$ implies $\mathsf{levT}(b) \leq \mathsf{levT}(c)$.

The reasoning procedure realizes a tight coupling between the concrete domain reasoner and the tableau algorithm: if the concrete domain reasoner finds that two concrete nodes are equal, the tableau algorithm may use this to deduce (via the computation of $\approx_{\mathsf{a}}$ and $\approx_c$) even more equalities between concrete nodes. The concrete domain reasoner may then return further "equalities" $\sim$ and so forth.

A tableau algorithm for $\mathcal{SHOQK}(\mathcal{D})$ is designed as a combination of the one for $\mathcal{SHOQ}(\mathbf{D})$ [83] and the one for $\mathcal{ALCOK}(\mathcal{D})$. To prove the termination of the algorithm for $\mathcal{ALCOK}(\mathcal{D})$-concepts, the authors introduced the notion of *role depth* of concepts. The most important difference of the algorithm for $\mathcal{SHOQK}(\mathcal{D})$ from the one for $\mathcal{ALCOK}(\mathcal{D})$ is that concepts of positive role depth can be added to arbitrary nodes in the completion forest[10]. Furthermore, the depth of a node

[8]Only boolean combinations of concept names appear in a Boolean KBox

[9]Only functional concrete roles appear in a path-free KBox

[10]A forest is generated because some completion rules remove nodes and edges from the completion tree.

might change because a node might become a root node due to some completion rules. Thus the role depth does not automatically decrease with the depth of nodes in the forest (as in the case of $\mathcal{ALCOK}(\mathcal{D})$) and a naive tableau algorithm would construct infinite forests. To enforce termination artificially, the relation $\prec$ is no longer required to respect the level of a node as in the $\mathcal{ALCOK}(\mathcal{D})$ case. A new node $x$ is introduced into the completion forest such that $y \prec x$ for all already existing nodes $y$.

The authors introduced a subset blocking mechanism in which they define the reflexive closure $\preceq$ of $\prec$. Unlike the subset blocking presented in [87], the blocking node in their solution is not necessarily an ancestor of the blocked node, but can be anywhere in the forest. The blocked nodes may even have unblocked successors. Their modification is explained as for obtaining a NExpTime upper bound.

Stating that tableaux are only used in proof, the authors introduced a tableau for $\mathcal{SHOQK}(\mathcal{D})$ that includes a function mapping concrete predicates used in the input concept and KBox to sets of values over the concrete domain. By this way, they argued that a concrete domain reasoner that can decide on the satisfiability of the predicate conjunctions built for tableaux is not necessary. However, to build such a tableau, the authors used a kind of concrete domain reasoner called $\mathcal{D}$-tester that verifies the satisfiability of those conjunctions and returns the equivalence of concrete nodes on forests.

The equivalence relation $\sim$ on concrete nodes, which is returned by $\mathcal{D}$-tester, is still used to compute the relation $\approx_{\mathsf{a}}$ that is then used by the tableau algorithm for $\mathcal{SHOQK}(\mathcal{D})$. However, because KBoxes are path-free, it is not necessary to compute the relation $\approx_c$ from $\approx_{\mathsf{a}}$ as in the $\mathcal{ALCOK}(\mathcal{D})$ case. $\mathcal{D}$-tester is called each time that the $R\exists c$ rule is applied (because this rule requires the update of the relation $\sim$).

There is no merging in their tableau algorithm. All the nodes in an equivalence class are assigned the same label. This is realized by choosing one representative whose node label contains the labels of all other nodes in the class and then copying the labels of all other nodes in the class from the representative label. The representative is the $\prec$-minimal node of the equivalence class.

Some main differences of our solution from the work presented in [97] are:

- We do not use identification constraints to define the equivalence relation on a graph, neither on abstract nodes nor on concrete nodes. We introduce a rule that verifies whether two nodes on a graph satisfy an IC in KBox, and merges them if they are identified by the same identifying set.

- We do not have to deal with the representative of nodes in an equivalence class. The reason is that by merging, there is no longer equivalent nodes on the graph.
- We have to deal with the infinite model property of $\mathcal{SHOIQK}(\mathbf{D})$, which provokes the problems that do not exist in $\mathcal{SHOQK}(\mathcal{D})$.
- We do not employ many notions applied in [97], such as the depth of abstract nodes, role depth, ordering of all abstract nodes, to enforce termination. Instead, we extend the notion of level of nominal nodes and of the priority of rule applications for $\mathcal{SHOIQ}$ to obtain termination.

## 5.6 Conclusion

In this chapter, we have introduced a reasoning procedure for the web ontology language OWL-K. Our decidable reasoning will provide utilities that ontology users and developers are expecting. In particular, it provides the capability to integrate data sources from relational databases into the Semantic Web environment through the ontology language OWL-K. It helps to verify the consistency of databases and ontologies in the design and development and to retrieve the semantics of data in the exploiting phase.

We do not choose to make a unique name assumption for our algorithm. However, it can easily be adapted to the unique name case by a suitable initialisation of the inequality relation. We do not address the data type reasoner and assume that it has an algorithm satisfying the requirement of our Tableau algorithm. As mentioned in Chapter 4, roles used in ICs are not required to be functional roles[11]. Actually, only individuals identified uniquely by those roles must be related to at most one other individual by each of those roles. Therefore, checking IC satisfiability, including number restrictions, is of a local nature and applied only to some nodes in the graph.

We have designed an algorithm that behaves like "pay as you go":

- if KBox is empty then the algorithm is treated as for $\mathcal{SHOIQ}(\mathbf{D})$;

[11]A role $R$ is called a functional role iff $\top \sqsubseteq \leq 1R$.

- if no condition of ICs is satisfied then the expansion rules for ICs are not applied;
- if no data types are used, the algorithm is treated as for $\mathcal{SHOIQK}$;
- if neither data types nor ICs are asserted, the algorithm is treated as for $\mathcal{SHOIQ}$.

Consequently, the performance of $\mathcal{SHOIQK}(\mathbf{D})$ is comparable with that of $\mathcal{SHOIQ}$, given that $\mathcal{SHOIQ}$ is NExpTime-complete [84]. However, regarding concrete domains and ICs, we have to leave the complexity of the algorithm as an open problem. If there exists an algorithm for $\mathcal{QIK}$-conformity in NP, then given the bound of number of nodes proved above (cf. the proof for Lemma 5.4.1), it is not hard to prove that our algorithm is 2-NExpTime-complete, but it is not obvious whether NExpTime-complete is also enough. Some results in this chapter have previously been published in [106].

CHAPTER 6

# Modeling ORM schemas in OWL-K

In the previous chapters, we have built OWL-K, a web ontology language with an underlying decidable description logic. OWL-K layers on top of OWL DL, and extends it with identification constraints. The latter is one of the most important factor in conceptualizing, both for relational databases and semantic web ontologies. Therefore, we consider OWL-K as a fundamental web ontology language to capture information from relational data sources through their conceptual schemas.

In this chapter, we will study a mechanism to model ORM schemas in OWL-K. The goal of this modeling is to integrate relational data sources modeled in ORM into the Semantic Web environment. In modeling ORM schemas, we mention also "OWL" (without "K"). It means that the transformation can be performed using OWL DL (the sub language of OWL-K). We will analyse data modeling in ORM, and try to represent ORM schemas in OWL(-K) ontologies. Besides presenting RDF/XML syntax, which makes ORM schemas usable in the Semantic Web, we will also introduce OWL-K abstract and DL syntax of ORM schemas, showing that the semantics of ORM diagrams are guaranteed.

The chapter is organized as follows. Section 6.1 briefly introduces how to model data in ORM. Consequently, we can identify the scope of our work in modeling ORM schemas in OWL-K. In Section 6.2, Section 6.3 and Section 6.4, we show how components in ORM schemas can be formalized in this web ontology language. Section 6.5 presents an algorithm to perform the translation of an ORM schema in an OWL-K ontology. Section 6.6 concludes the chapter.

## 6.1 Data Modeling in ORM

Understanding data modeling in ORM is necessary for modeling ORM schemas in OWL-K. In Chapter 2, we have shown the main features of ORM that distinguish ORM from ER and UML methods. However, to understand how ORM performs data modeling, more explanation is needed. This section will talk about that. Consequently, we will identify the scope of the transformation of ORM into OWL-K. Full description of ORM methodology can be found in [67, 64].

In the 1970s, substantial research was carried out to provide high level semantics for modeling information systems. Falkenberg introduced the fundamental ORM framework, which was called the "object-role model" [50]. This framework allowed n-ary and nested relationships, but depicted roles with arrowed lines. Nijssen adapted this framework by introducing the circle-box notation for objects and roles that has become standard, and adding a linguistic orientation and design procedure to provide a modeling method called ENALIM (Evolving NAtural Language Information Model) [109, 110]. Other researchers extended the method with the semantics [91], subtypes, and an ORM query language (RIDL) [102]. The method was renamed "Nijssen Information Analysis Method" (NIAM). In later years the acronym "NIAM" was given different expansions, and is now known as "Natural language Information Analysis Method". Halpin, then, provided the first full formalization of the method [65], including schema equivalence proofs, and made several refinements and extensions to the method.

Many researchers have contributed to the ORM method over the years. Today various versions of the method exist, but all adhere to the fundamental object-role framework (e.g. [118, 61, 129, 132, 48, 10]). In this thesis, we adopt the ORM 2, the second generation of ORM developed by Halpin [69] and supported by many tools (see Chapter 7).

### 6.1.1 Conceptual Schema Design Procedure

ORM focuses on data modeling, since the data perspective is the most stable and it provides a formal foundation on which operations can be defined. For correctness, clarity and adaptability, information systems are best specified first at the conceptual level, using concepts and language that people can readily understand. Analysis and design involves building a formal model of the application area or *universe of discourse* (UoD). To do this properly requires a good understanding of the UoD and a means of specifying this understanding in a clear, unambiguous way. ORM simplifies this process by using natural language, as well as intuitive diagrams that can be populated with examples, and by expressing the information in terms of elementary relationships.

The ORM's conceptual schema design procedure (CSDP) specifies the information structure of the application by identifying the *types of fact* that are of interest; *constraints* on these; and perhaps *derivation rules* for deriving some facts from others. With large applications, the UoD (i.e. the application domain) can be divided into convenient modules, the CSDP is then applied to each, and the resulting subschemas are integrated into the global conceptual schema.

Table 6.1 shows the steps in a CSDP. *Step 1* is the most important. Examples of the information required from the system are verbalized in natural language, such as in the form of output reports or manual versions of the required system. To avoid misinterpretation, a UoD expert should perform or at least check the verbalization. *Step 2* is to draw a draft diagram of the fact types and apply a population check. As a check, each fact type has been populated with at least one fact, shown as a row of entries in the associated fact table. It is useful for validating the model with the client and for understanding constraints. However, the sample population is not part of the conceptual schema itself. *Step 3* checks for entity types and may combine several types such that the original information about them is preserved. Another aspect of Step 3 is to see if some fact types can be derived from others by arithmetic. In this case, a *derivation rule* must be supplied below the schema diagram. Note that this rule is not included in the ORM diagram. The other steps are devoted to adding constraints such that the diagram is always consistent.

Table 6.1: The conceptual schema design procedure

| Step | |
|---|---|
| 1. | Transform familiar information examples into elementary facts, apply quality checks. |
| 2. | Draw the fact types, apply a population check. |
| 3. | Check for entity types that should be combined, note any arithmetic derivations. |
| 4. | Add uniqueness constraints. |
| 5. | Add mandatory constraints, check for logical derivations. |
| 6. | Add value, set comparison and subtyping constraints. |
| 7. | Add other constraints and perform final checks. |

Once the procedure is finished, the global schema is mapped to a relational database schema by conceptual schema transformations (see Chapter 7).

### 6.1.2 ORM Diagrams

As mentioned in the previous section, an ORM conceptual schema is depicted by a diagram. An ORM diagram is built using information about relations between object types or, in other words, about facts. In ORM, a fact is always *elementary*. An elementary fact asserts that a particular object has a property, or that one or more objects participate in a relationship, where that relationship cannot be expressed as a conjunction of simpler (or shorter) facts.

A fact comprises objects and the roles the objects play. All the roles in a fact together make up the predicate of that fact. The great majority of facts contain two roles. To illustrate the anatomy of a fact, we revisit the example in Chapter 2:

*The Department with the ID "IT" employs / works for the Employee with the name "Hang DO".*

In an ORM diagram, you would symbolize this fact using the notation shown in Figure 6.1 and explained below.

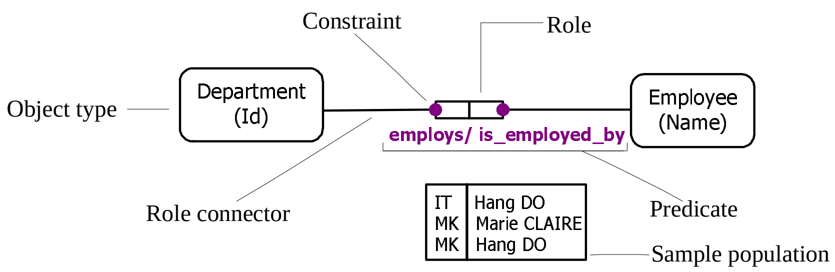


Figure 6.1: Symbolizing a fact

An *object type*, which appears as a soft rectangle (either solid or dashed), categorizes data into different kinds of meaningful sets. For example, the Department object type divides the application domain into those things that are departments (or department IDs) and those that are not. A thing or noun in the application domain will always be an object type in ORM.

A *predicate* is a verb, or verb phrase, that connects the object types in a fact type. A predicate provides the semantic context for objects and consists of one or more roles that objects play.

Each *role* in a predicate is expressed by a *role box* and is played by one object type. In Figure 6.1, the left role (role position 1 in the predicate) is *employs*. What is being said is that a department employs an employee. Or you might say, "department is playing the role of *employing* an employee". At the right role position (role position 2), the Employee object type is playing a role with respect to the Department object type. If you read the predicate from right to left, you can say, "employee is employed by department". The role *is employed by* is separated from the role *employs* in the predicate by a slash.

A solid dot at the role end signifies a *constraint* over that role position. In this case, the dot represents a mandatory constraint.

*Sample population* provides examples of allowable values for the object types at their respective role positions. Sample data in each row represent a valid instance (or fact) for a fact type. Each column is associated with a role that an object plays in the fact. The set of instances in a column represents the legal values for the fact at that column or role position.

A fact type defines the set of all possible instances of objects that play roles within a fact. Actually, ORM diagrams depict fact types and constraints on them. As shown in the above example, Department employs Employee is a fact type. It defines the set of allowable values (via its constraints) of the object types that play roles within it. *Department "IT" employs Employee "Hang DO"* is just a single fact instance, or an example, of the data allowed for the fact type. Department employs Employee could also have instances such as the following:

*Department "MK" employs Employee "Marie CLAIRE"*

*Department "MK" employs Employee "Hang DO"*

Figure 6.2 summarizes the main symbols used in modeling data in ORM. A value type is usually shown as a *named, dashed soft rectangle* (symbol 1). Another notation for value types encloses the value type name in parentheses (cf. symbol 4). An entity type is depicted as a *named solid soft rectangle* (symbol 2). In symbol 3, an *exclamation mark* is added to declare that an entity type is independent. The reference scheme may be abbreviated as in symbol 4 by displaying the reference mode in parentheses beside the name of the entity type, e.g. Country(code). Value constraints are described by a list next to the object type. Symbol 6 shows a binary predicate, comprised of two roles. If we want to talk about a relationship type we may objectify it (i.e. make an object out of it) so that it can play roles. Graphically, the objectified predicate is enclosed in either a soft rectangle (symbol 7) or an ellipse, and named. Objectified predicates are also called nested object types. A solid arrow (symbol 8) denotes the subtype constraint. A mandatory role

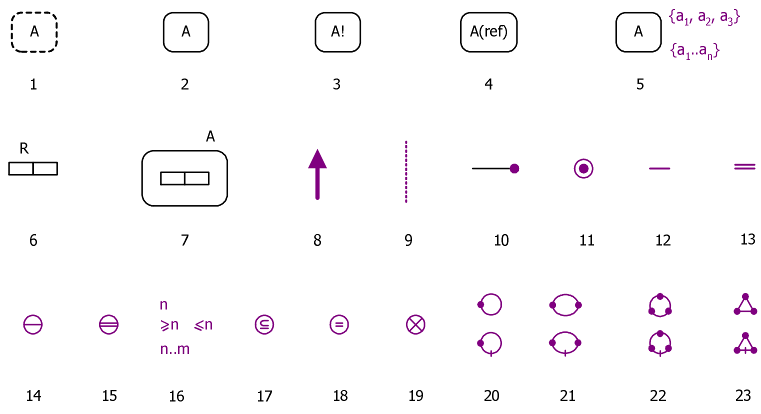


Figure 6.2: Main symbols in ORM diagrams

constraint is usually shown as a *solid dot* (symbol 10). A disjunctive mandatory constraint may be applied to two or more roles. This may often be shown by connecting the roles by dotted lines (symbol 9) to a circled solid dot (symbol 11). *Internal uniqueness* constraints are depicted as *simple lines* (symbol 12). A predicate may have one or more uniqueness constraints, at most one of which may be declared primary (i.e. preferred) by adding a *double line* (symbol 13). An external uniqueness constraint shown as a *circled underline* (symbol 14) may be applied to two or more roles from different predicates by connecting to them with dotted lines. To declare an external uniqueness constraint primary, we use a *circled double underline* instead of circled underline (symbol 15). Symbol 16 shows four kinds of frequency constraint. Applied to a sequence of one or more roles, these constraints indicate that instances that play those roles must do so exactly $n$ times (symbol 16, top), at least or at most $n$ times (symbol 16, middle), or between $n$ and $m$ times (symbol 16, bottom). Symbols 17-19 denote set comparison constraints, and may only be applied between compatible role sequences. A circled "$\subseteq$" (symbol 17) is a subset constraint. A circled "$=$" (symbol 18) is an equality constraint. A circled "$\times$" (symbol 19) is an exclusion constraint. Symbol 20-23 shows four pairs of ring constraint, that may be applied to a pair of roles played by the same host type. Each pair depicts two kinds of constraint one of which negates the other.

In the following sections, we will try to model ORM schemas described by using the above symbols into OWL-K. We do not address derivation rules in this thesis because as mentioned in the previous section, these rules are actually not a part of ORM diagrams. We limit our translation to ORM in binary version, i.e., only unary and binary predicates are appeared in ORM diagrams. Note that a relational database schema is actually binary (binary relationships between a relation and an attribute). Therefore, the binary version of ORM is enough for modeling relational data.

## 6.2 Basic Components

The fundamental components of ORM are object types, which consist of entities and values, and roles. Nil is never depicted in an ORM diagram. That is, object types are non-nil. Object types are connected by the roles they play, composing the fact types. All the roles in a fact type make up its predicate whose arity is the number of roles. In this section we describe the nature of object types, predicates and roles, and provide their transformation into OWL-K.

### 6.2.1 Value Types

A *value object type* (or *value type* for short) is an object type that cannot be defined by other object types. For example, Lastname is a value type, which is illustrated in an ORM diagram as in Figure 6.3. There are two kinds of values, namely strings and numbers. A value is a string if it has no arithmetic significance. For example, the *LastName* “Smith” is just the string of characters S-m-i-t-h. A value is a number if the value represented by the symbol can be operated on mathematically. For example, in the fact type *Employee hasAge YearOld*, *YearOld* would be represented by a number.

Figure 6.3: A value type represented in an ORM diagram

Value type is used to describe entities. For example, you could never find the *LastName* “Smith”, but you could certainly find a *Employee* having the last name “Smith”.

If a value type is represented as a class in OWL, then the value type *Lastname* will be defined in OWL as the following:

```
<owl:Class rdf:ID = "Lastname"/>
```

In this case, no class representing a value type should be appeared on the left hand side of an axiom, i.e., be defined by others in an axiom. For example, the class Lastname defined above should never be a subclass of another. However, the above class definition example in OWL shows that this feature cannot be expressed explicitly (e.g. as a primitive class or as a class having no subclass). With this expression, we know only the existence of the class while in ORM diagram, we can easily find that Lastname is the primitive by the figure that illustrates it. Since an ontology is an open knowledge base, a class may no longer be primitive if an ontology extension is performed (e.g. adding to an ontology an axiom whose left-hand side includes a class that has never appeared on that side before). Thus, the semantics of the original part (w.r.t the extended part) of an ontology may have been changed. Consequently, the ontology may no longer be consistent. In this case, there should have a mechanism to verify the consistency of ontologies representing ORM diagrams and to ensure that the semantics of ontologies fully capture the original semantics expressed in ORM diagrams. Moreover, representing value types by classes does not capture the string or number property of value types. Nevertheless, instances of value types are actually strings or numbers. And a data type is not defined by any object type. Therefore, we translate value types as data types rather than classes in OWL (see Figure 6.4).

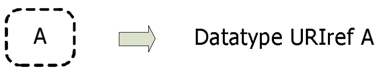


Figure 6.4: VL-rule - Modeling value types in OWL

A value type will be represented as a new simple data type derived from the built-in XML Schema data type *string* or *integer*. For example, the value type *Lastname* above can be defined as follows:

```
<xsd:simpleType name = "Lastname">
  <xsd:restriction base = "xsd:string">
  <xsd:maxLength value = "30"/>
 </xsd:restriction>
</xsd:simpleType>
```

### 6.2.2 Entity Types

An *entity object type* (or *entity type* for short) is an object type representing a conceptual thing in the real world. Tangible objects, such as `Employee`, and intangible concepts, such as `Subject`, are both considered entity types. Essentially, an entity type is a concept or, in other words, a class in OWL. As mentioned above, an entity type can also be declared independently by using an exclamation mark in graphical notation. This means that instances of that type may exist without participating in any facts. By default, it is not the case, i.e., object is not introduced into the universe if it does not take part in some fact. In an OWL ontology, we can declare any class without concerning its participation in some axiom. Thus such an entity type obviously corresponds to a class in OWL. We summarize the modeling of entity types in Figure 6.5.

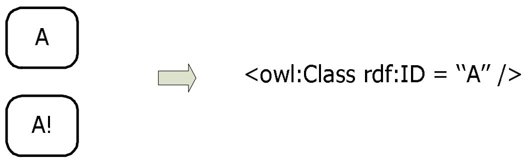


Figure 6.5: EN-rule - Modeling entity types in OWL

To give an example, the entity type Employee above will be represented in an ORM diagram (on the left) and in OWL (on the right) as shown in Figure 6.6.

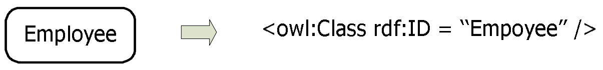


Figure 6.6: Example of modeling entity types in OWL

An entity type can be described by other entity or value types, or by a composition of different object types. When identified by a single value type, entity type is described using the reference mode (cf. symbol 4 in Figure 6.2). There are two other kinds of entity type: nested (cf. symbol 7 in Figure 6.2) and composite. All these descriptions of entity type must use constraints. Therefore, we postpone the translation of these types until Section 6.4.

### 6.2.3 Predicates and Roles

A predicate is the part of a fact type that describes the roles the objects play. A predicate of arity $n$ is displayed as a named, contiguous sequence of $n$ boxes, where $n \geq 1$. Figure 6.7 shows some predicate examples. Each box depicts a role which can be named or not. The name of a role is written in or beside the relevant box. Predicates are normally treated as ordered. In this case, the name of a predicate is usually the ordered list of names of its role components. It means that different readings may be provided so the information may be read in any direction. For example, in the fact type Department employs/works for Employee, the predicate name is employs/works for and we can read from left to right as "Department employs Employee", or from right to left as "Employee works for Department". By default, predicates are read left-to-right and top-down; prepending "<<" to a predicate reading reverses the reading order.

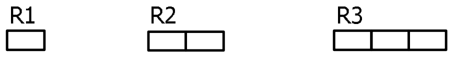
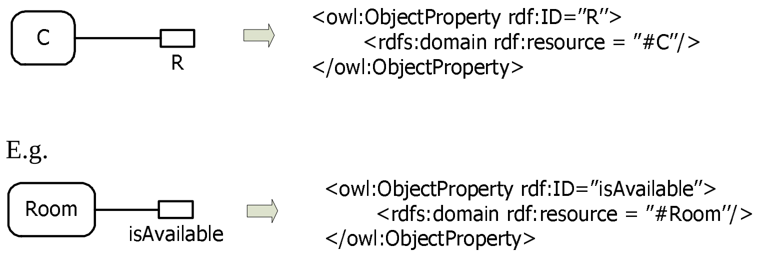


Figure 6.7: Unary, binary and ternary predicates in an ORM diagram

In an ORM diagram, each role box must be connected by a role connector to *exactly one* soft rectangle. Moreover, each predicate name appears *only once*. Informally, each role is designed so that only objects satisfying the predicate of the attached rectangle play that role. A fact type receives its arity from the arity of its predicate. That is, if the predicate of a fact type is unary (binary), then the fact type is unary (binary).

*Unary Fact Types.* A unary fact type represents a property of an object type. For example, one states that Room is available to describe rooms which are available. Note that in OWL, ranges are not necessary to be specified for properties. Consequently, roles in unary fact types can be translated into an object property in OWL whose domain is the object type (cf. Figure 6.8).



```
<owl:ObjectProperty rdf:ID="R">
      <rdfs:domain rdf:resource = "#C"/>
</owl:ObjectProperty>
```

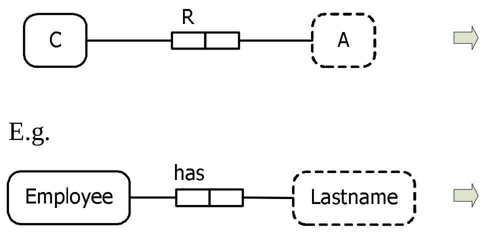
```
<owl:ObjectProperty rdf:ID="isAvailable">
      <rdfs:domain rdf:resource = "#Room"/>
</owl:ObjectProperty>
```

Figure 6.8: R0-rule - Modeling roles in unary predicates in OWL

A value type would not play the role in a unary fact type. The reason is that neither a string nor a number can be identified as a single instance, and that you could not identify an instance of a string or number to play the role.

*Binary Fact Types.* Binary fact types are by far the most common. A role in a binary predicate goes always with two object types one of which plays the role with the other. For example the role represents is played by the object type Student with the object type Country in the fact type Student represents Country. Therefore, an OWL property representing such a role should have the domain specified by the object type playing the role, and the range specified by the object type subjected to by the role.

A role can be attributive, i.e., it can define a value of an entity type. For example, in the fact type `Employee has Lastname`, where Employee is an entity type and Lastname is a value type, has is an attributive role. Since value types are represented as data types in OWL, an attributive role corresponds to a data type property in OWL. Figure 6.9 shows the translation of attributive roles into OWL.

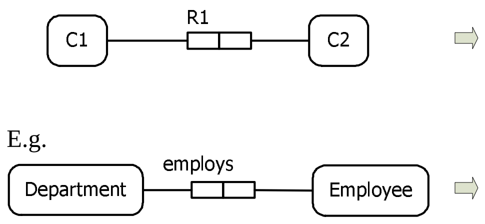


```
<owl:DatatypeProperty rdf:ID="R">
      <rdfs:domain rdf:resource = "#C"/>
      <rdfs:range rdf:resource = "#A"/>
</owl:DataypeProperty>

<owl:DatatypeProperty rdf:ID="has">
      <rdfs:domain rdf:resource = "#Employee"/>
      <rdfs:domain rdf:resource = "#Lastname"/>
</owl:DatatypeProperty>
```

Figure 6.9: R1-rule - Modeling attributive roles in OWL

Otherwise, a role connects one entity to another (in case of binary predicate). In this case, the role will be translated into object property in OWL. Revisiting the fact type `Department employs Employee`, where both Department and Employee are entity types, we see that the role employs corresponds to an object property in OWL. Figure 6.10 shows the rule for this translation.

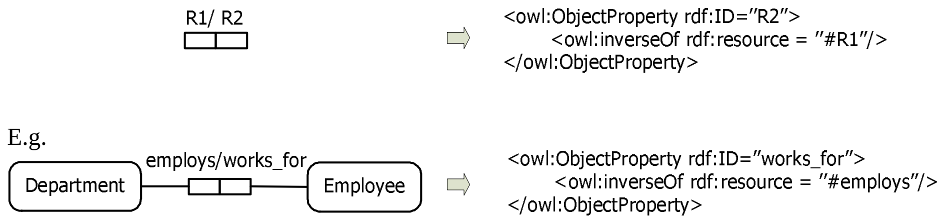


```
<owl:ObjectProperty rdf:ID="R1">
      <rdfs:domain rdf:resource = "#C1"/>
      <rdfs:range rdf:resource = "#C2"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="employs">
      <rdfs:domain rdf:resource = "#Department"/>
      <rdfs:range rdf:resource = "#Employee"/>
</owl:ObjectProperty>
```

Figure 6.10: R2-rule - Modeling non-attributive roles in OWL

Besides, there may exist two role names in a binary predicate. For example, in the fact type `Department employs/works_for Employee`, employs and works_for are two role names in the predicate. Actually, the role in position 1 of the predicate describes the relationship of the object that plays the role to the object type subjected to by the role, while the role in position 2 of the predicate describes the same relationship but in the inverse order. That is, employs describes the relationship of the entity type Department to the entity type Employee while works_for describes the same relationship but from Employee to Department. Hence, the role in position 2 is actually the inverse role of the role in position 1 and vice versa. Therefore, in case a binary predicate comprises two predefined non-attributive roles, one role is mapped as an object property as shown above while the other is mapped as the inverse of it. Figure 6.11 shows the translation for inverse roles.



```
<owl:ObjectProperty rdf:ID="R2">
      <owl:inverseOf rdf:resource = "#R1"/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="works_for">
      <owl:inverseOf rdf:resource = "#employs"/>
</owl:ObjectProperty>
```

Figure 6.11: IR-rule - Modeling inverse roles in OWL

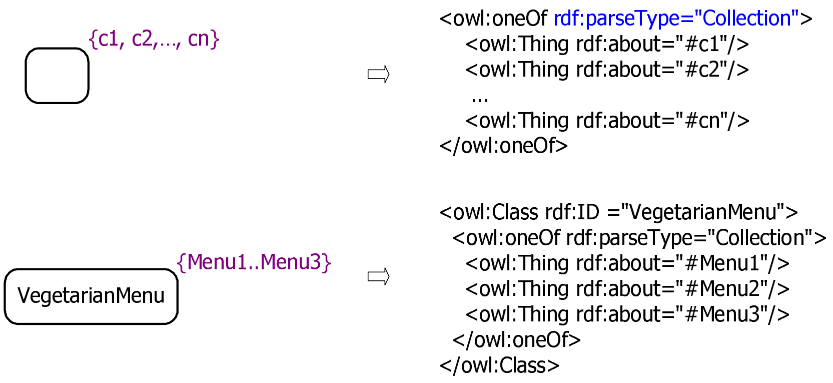
## 6.3 Object Constraints

ORM introduces some constraints that can be set on object types. In this section, we will discuss these constraints and show their translation into OWL. Constraints that apply to roles will be dealt with in Section 6.4.

### 6.3.1 Value Constraints

Value constraints (VCs) specify a list of possible values for an object type. To restrict an object type's population to a given list, the relevant values may be listed in braces (cf. on the right top of symbol 5 in Figure 6.2). If the values are ordered, a range may be declared separating the first and last values by ".." (cf. on the right bottom of symbol 5 in Figure 6.2). For example, an entity type VegetarianMenu is restricted to Menu1..Menu3 and a value type Sex is restricted to 'M', 'F'. Open ranges are also supported (e.g. "> 0" for InterestRate).

OWL provides an exhaustive enumeration of individuals that together form the instances of a class. This kind of class is defined with the owl:oneOf property whose value must be a list of individuals that are the instances of the class. Since we translate entity types into classes in OWL, value constraints on entity types can be translated into OWL using the owl:oneOf constructor. In OWL a list of individuals must be represented explicitly, hence all elements of a range of value of an entity type must be stated. For example, the entity type VegetarianMenu above should be listed completely as a set of Menu1, Menu2, and Menu3. The rule of translation of VCs on entity types and its application to the VC on VegetarianMenu are shown in Figure 6.12.



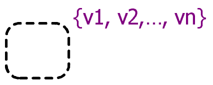
```
<owl:oneOf rdf:parseType="Collection">
   <owl:Thing rdf:about="#c1"/>
   <owl:Thing rdf:about="#c2"/>
    ...
   <owl:Thing rdf:about="#cn"/>
</owl:oneOf>
```

```
<owl:Class rdf:ID ="VegetarianMenu">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Menu1"/>
    <owl:Thing rdf:about="#Menu2"/>
    <owl:Thing rdf:about="#Menu3"/>
  </owl:oneOf>
</owl:Class>
```

Figure 6.12: VC1-rule - Modeling VCs on entity types in OWL

For VCs on value types, we have translated value types into datatypes in OWL. Besides, OWL provides also owl:oneOf to define an enumeration of data values, namely an enumerated datatype. Consequently, we will use owl:oneOf to translate the VCs in the form of enumeration of value types. As for the case of translation for VCs on entity types, owl:oneOf for datatype does not support the VCs in the form of range. Therefore, all the elements in a closed range must be listed explicitly. Note that in the case of an enumerated datatype, we cannot use rdf:parseType="Collection" as for the translation of VCs on entity types, but the basic list constructors rdf:first, rdf:rest and rdf:nil. Figure 6.13 shows the detailed translation for this kind of constraint and an application to the value type "Sex" above. In Figure 6.13, we use "xmltype" to denote "&xsd;integer" or "&xsd;string". Note that in OWL, an enumerated datatype can only be defined in a data type property definition.

An open range of a value type actually limits the number of possible values for that type, based on the string or number type. Therefore, this constraint can be translated as a derivation by restriction on an existing type, which is actually considered as a new simple data type derived from the built-in XML Schema data type string or number. As presented in Chapter 4, this kind of derivation is supported by OWL. Therefore, we apply the VL-rule to this constraint. That is, open range value constraints on value types will be translated into new simple

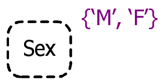


⇨

```
<owl:oneOf>
       <rdf:List>
          <rdf:first rdf:datatype="xmltype">v1</rdf:first>
          <rdf:rest>
            <rdf:List>
              <rdf:first rdf:datatype="xmltype">v2</rdf:first>
               ...

                 <rdf:first rdf:datatype="xmltype">vn</rdf:first>
                 <rdf:rest rdf:resource="&rdf;nil" />
           </rdf:List>
         </rdf:rest>
       </rdf:List>
</owl:oneOf>
```

E.g.



⇨

```
<owl:DataRange rdf:ID = "Sex">
   <owl:oneOf>
      <rdf:List>
         <rdf:first rdf:datatype="&xsd;string">M</rdf:first>
         <rdf:rest>
           <rdf:List>
             <rdf:first rdf:datatype="&xsd;string">F</rdf:first>
             <rdf:rest rdf:resource="&rdf;nil" />
           </rdf:List>
         </rdf:rest>
      </rdf:List>
   </owl:oneOf>
</owl:DataRange>
```

Figure 6.13: VC2-rule - Modeling enumerated VCs on value types in OWL

data types derived from the built-in XML Schema data type string or integer. For example, "> 0" on value type InterestRate will be translated as

```
<xsd:simpleType name = "InterestRate">
  <xsd:restriction base = "xsd:decimal">
  <xsd:minExclusive value = "0.00"/>
 </xsd:restriction>
</xsd:simpleType>
```

## 6.3.2 Subtypes and Supertypes

A subtype is a special kind of object type. It is an entity object type whose population represents a portion of the population of another entity type, called the supertype. For example, Employee can be a subtype of a supertype Person. Every supertype and subtype must play at least one role. A subtype is displayed graphically by first representing it as a distinct object type, then drawing an arrow from this rectangle to the rectangle that represents the supertype (cf. Figure 6.14).

Actually, this arrow is a constraint that links the two entity types.

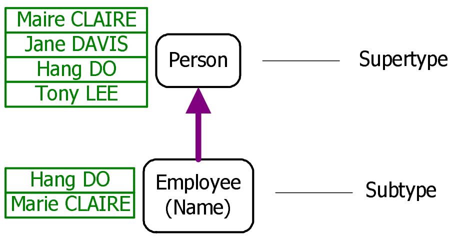
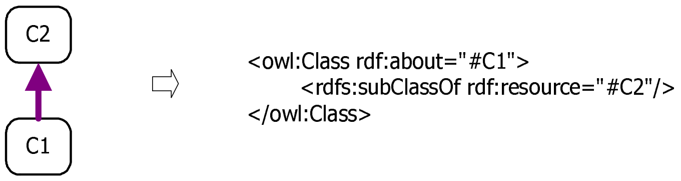


Figure 6.14: Modeling derived predicates with one asterisk in OWL

An entity type is translated into a class in OWL, hence subtype corresponds actually to subclass in OWL. Thus, we use the rdfs:subClassOf constructor to translate this kind of constraint, as shown in Figure 6.15.



```
<owl:Class rdf:about="#C1">
      <rdfs:subClassOf rdf:resource="#C2"/>
</owl:Class>
```

Figure 6.15: ST-rule - Modeling subtypes and supertypes in OWL

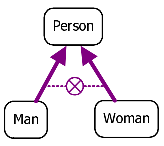
### 6.3.3 Exclusion and Exhaustion Constraints

By default, ORM subtypes may overlap. ORM allows graphic constraints to be added to indicate that subtypes are mutually exclusive, collectively exhaustive, or both.

Two types are mutually exclusive if an instance of one type is not simultaneously an instance of the other type. For example, Figure 6.16(a) describes two object types Man and Woman that are subtypes of Person and are mutually exclusive (by using a circled "×" connected to the relevant subtypes via dotted lines). In OWL, we can use the owl:disjointWith constructor to express this exclusion. Hence, the exclusion example above can be translated into OWL as shown in Figure 6.16(b). The translation rule generalized for the exclusion constraint on n subtypes can be seen in Figure 6.16(c).

Two subtypes are collectively exhaustive if the union of their populations exhausts the population of their supertype. For example, in the example above, Person must be either Man or Woman. We say that the union of the populations of the subtypes

E.g.



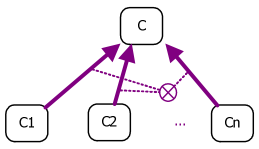
(a)

⇨

```
<owl:Class rdf:about="#Man">
  <owl:disjointWith rdf:resource="#Woman"/>
</owl:Class>
```

(b)

ME-rule:



⇨

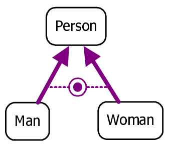
```
<owl:Class rdf:about="#C1">
    <owl:disjointWith rdf:resource="#C2"/>
     ...
    <owl:disjointWith rdf:resource="#Cn"/>
</owl:Class>
<owl:Class rdf:about="#C2">
    <owl:disjointWith rdf:resource="#C3"/>
     ...
    <owl:disjointWith rdf:resource="#Cn"/>
</owl:Class>
...
<owl:Class rdf:about="#Cn-1">
    <owl:disjointWith rdf:resource="#Cn"/>
</owl:Class>
```

(c)

Figure 6.16: Modeling the mutual exclusion constraint in OWL

Man and Woman exhausts the population of the supertype Person and depict this relation in ORM diagram as in Figure 6.17(a) (using a circled dot). In OWL, we can use owl:unionOf constructor to describe the constraint, so the exhaustion example in Figure 6.17(a) can be represented in OWL as in Figure 6.17(b). The translation rule generalized for the exhaustion constraint on n subtypes can be seen in Figure 6.17(c).

E.g.



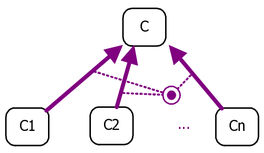
(a)

⇨

```
<owl:Class rdf:about="#Person">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Man" />
    <owl:Class rdf:about="#Woman" />
  </owl:unionOf>
</owl:Class>
```

(b)

CE-rule:



⇨

```
<owl:Class rdf:about="#C">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#C1" />
     ...
    <owl:Class rdf:about="#Cn" />
  </owl:unionOf>
</owl:Class>
```

(c)

Figure 6.17: Modeling the collective exhaustion constraint in OWL

When both exclusion and exhaustion constraints need to be applied (e.g. the case of subtypes Man and Woman of supertype Person), in ORM diagram we can use a single symbol (a circled and crossed dot) instead of the two symbols presented above.

## 6.4 Role Constraints

Role constraints are rules that govern the population of a database and restrict the data allowed in the database. A constraint can be applied to the population for one or more objects that play a single role, to a role sequence, i.e. a group of roles in one fact type, or to a set of role sequences in many fact types. The number of objects and roles involved in a constraint depends on the purpose and type of the constraint. This section will describe how to translate fact types under constraints into OWL. It also describes the translation of object types with reference mode, composite and nested object types.

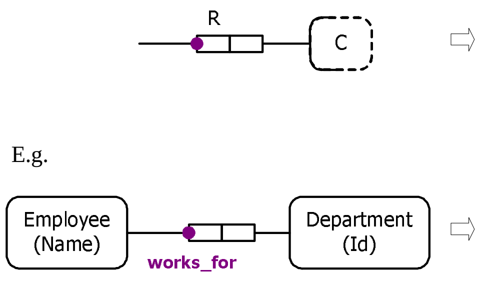
### 6.4.1 Mandatory Constraints

*Simple Mandatory Constraints. A mandatory constraint (MC)* on a role is often called a simple MC. It specifies that every instance of its object type must play the role. The object type subjected to by the role can be either a value type or an entity type. For example, a simple MC is applied on the role works for to show that every employee must work for some department.

In OWL, the value constraint owl:someValuesFrom is a built-in OWL property specifying that all individuals of the restricted class must be linked to at least one value of the property concerned. In other words, for each individual $x$ of the restricted class, there exists a $y$ (either an instance of the class description or value of the data range) such that the pair (x,y) is an instance of the concerned property.

MCs, therefore, can be expressed by the constraint owl:someValuesFrom in OWL. The translation rule MC2-rule for this kind of constraint and an example are shown in Figure 6.18. As mentioned in Section 6.1.2, in an ORM diagram a role is explicitly indicated as mandatory by adding a solid large dot at the role end of the role connector. In the figure, a soft rectangle that is half solid and half dashed is an abbreviation for any object type.

Note that the translation in Figure 6.18 is applied to only binary predicates. Besides, we can also have simple MCs applying to unary predicates. For example,

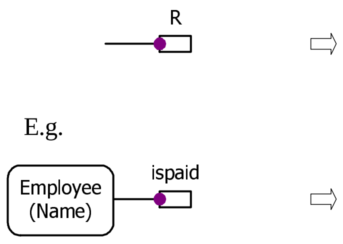


```
<owl:Restriction>
    <owl:onProperty rdf:resource="#R" />
    <owl:someValuesFrom rdf:resource="#C" />
</owl:Restriction>
```

```
<owl:Class rdf:about="#Employee">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#works_for" />
      <owl:someValuesFrom rdf:resource="#Department" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Figure 6.18: MC2-rule - Modeling MCs in binary predicates in OWL

it is obligatory that each employee is paid. In this case, we can also employ the constraint owl:someValuesFrom in OWL for the translation. Due to R0-rule, the range of the role in a unary predicate (while not specified) can be considered as the universe, or in other words, as owl:Thing in OWL. Therefore, we translate a simple MC applying to a role in a unary predicate as in Figure 6.19.



```
<owl:Restriction>
  <owl:onProperty rdf:resource="#R"/>
  <owl:someValuesFrom rdf:resource="&owl;Thing"/>
</owl:Restriction>
```

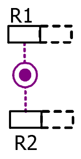
```
<owl:Class rdf:about="#Employee">
  <rdfs:subClassOf>
     <owl:Restriction>
        <owl:onProperty rdf:resource="#ispaid"/>
        <owl:someValuesFrom rdf:resource="&owl;Thing"/>
     </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Figure 6.19: MC1-rule - Modeling MCs in unary predicates in OWL

Note that no value type plays a mandatory role, because value type represents just strings or numbers some of which might never be used in an application.

*Disjunctive Mandatory (Inclusive-Or) Constraints.* A disjunctive mandatory constraint (DMC) is a mandatory constraint on a combination of two or more roles connected to the same object type. It specifies that each instance of an object type's population must occur in at least one of the constrained roles. For example, one would like to express that every Employee must be paid or be provided with house. Hence, we translate this constraint as a disjunction of all the simple MCs on the constrained roles respectively. In OWL, disjunction is described by the owl:unionOf property.

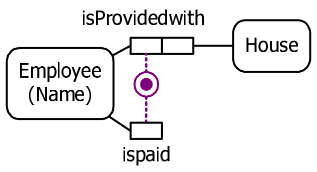
Figure 6.20 shows the translation of an DMC on two predicates in detail. In an ORM diagram, a DMC is depicted by placing the solid dot in a circle connected by dotted lines to the roles it applies to. In the figure, the predicate whose second role box is dashed is an abbreviation for either an unary or a binary predicate. We use this notation also in the following sections.



⇨

```
<owl:unionOf rdf:parseType="Collection">
   <Apply rule MC1 or rule MC2 on R1>
   <Apply rule MC1 or rule MC2 on R2>
</owl:unionOf>
```

E.g.

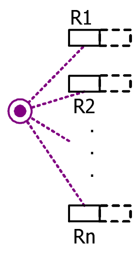


⇨

```
<owl:Class rdf:about="#Employee">
   <rdfs:subClassOf>
      <owl:Class>
         <owl:unionOf rdf:parseType="Collection">
            <owl:Restriction>
               <owl:onProperty rdf:resource="#isProvidedwith"/>
               <owl:someValuesFrom rdf:resource="#House"/>
            </owl:Restriction>
            <owl:Restriction>
               <owl:onProperty rdf:resource="#ispaid"/>
               <owl:someValuesFrom rdf:resource="&owl;Thing"/>
            </owl:Restriction>
         </owl:unionOf>
      </owl:Class>
   </rdfs:subClassOf>
</owl:Class>
```

Figure 6.20: Modeling DMCs on two predicates in OWL

The translation of DMC is generalized to the case of $n$ predicates as in Figure 6.21. The $n$ predicates including R1, ..., Rn respectively ($n \geq 2$) are either unary or binary.



⇨

```
<owl:unionOf rdf:parseType="Collection">
   <Apply rule MC1 or rule MC2 on R1>
   <Apply rule MC1 or rule MC2 on R2>
   ...
   <Apply rule MC1 or rule MC2 on Rn>
</owl:unionOf>
```

Figure 6.21: DMC-rule - Modeling DMCs in OWL

## 6.4.2 Internal Uniqueness Constraints

*Uniqueness constraints (UCs)* are used to express that each instance of an object type plays a set of roles at most once. A role sequence is a group of roles in one predicate. The set of roles under the constraint consists of one or many sequences of roles in different predicates. Therefore, UCs can be categorized into two kinds: internal uniqueness constraints (UCs on only one predicate) and external uniqueness constraints (UCs on many predicates). In this section, we talk about internal uniqueness constraints. External uniqueness constraints will be discussed in Section 6.4.5.

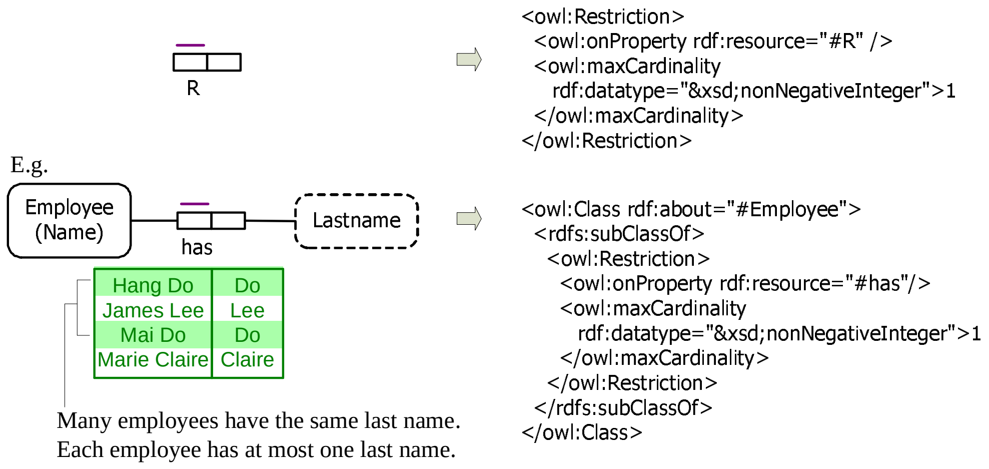
Internal uniqueness constraints (IUCs) are UCs on a single predicate. An internal uniqueness constraint is indicated by a simple line (cf. symbol 12 in Figure 6.2) over a role or a role sequence. The simple line spanning $n$ roles of a predicate ($n > 0$) means that the combination of $n$ columns of data in the respective table is unique. For example, an IUC on the role has in the fact type Employee has Lastname means that each Employee has at most one Lastname. In other words, each occurrence of an employee in the population is unique.

These constraints, therefore, make think of the maxCardinality restriction in OWL. The cardinality constraint owl:maxCardinality is a built-in OWL property that links a restriction class to a data value belonging to the value space of the XML Schema datatype nonNegativeInteger. A restriction containing an owl:maxCardinality constraint describes a class of all individuals that have at most $n$ semantically distinct values (individuals or data values) for the property concerned, where $n$ is the value of the cardinality constraint. Since IUCs restrict the number of occurrences of an object to only one, $n$ should be equal to one. IUCs do not apply to unary fact types because an instance cannot be repeated within the population of a unary fact type.

There are three possible ways (or patterns) to define data uniqueness for a binary fact type: many-to-one, one-to-one, and many-to-many.

### 6.4.2.1 Many-to-One

When an IUC is applied to a single role in a binary fact type, the role under the IUC cannot include duplicate values, while the other role can. We already saw this type of relationship with the fact type Employee has Lastname. The common term for describing this kind of relationship between two objects is "many-to-one". The constrained role is said to have a many-to-one relationship with the unconstrained role. For example, we can say "There is a many-to-one relationship between Employee and Lastname" because the same last name can occur with many employees, but one employee can have only one last name. This pattern has the direct translation into the maxCardinality described above. The detailed translation and an example can be seen in Figure 6.22. In the example, two employees (with the name "Hang Do" and "Mai Do") have the same last name ("Do").


R
E.g.
Employee (Name)
has
Lastname
Many employees have the same last name.
Each employee has at most one last name.

| | |
|---|---|
| Hang Do | Do |
| James Lee | Lee |
| Mai Do | Do |
| Marie Claire | Claire |

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#R" />
  <owl:maxCardinality
    rdf:datatype="&xsd;nonNegativeInteger">1
  </owl:maxCardinality>
</owl:Restriction>
```

```
<owl:Class rdf:about="#Employee">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#has"/>
      <owl:maxCardinality
        rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

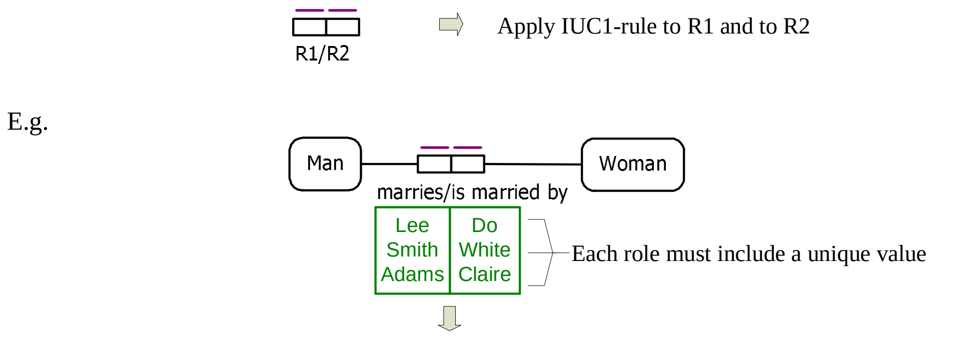
Figure 6.22: IUC1-rule - Modeling IUCs spaning one role in OWL

#### 6.4.2.2 One-to-One

Applying an internal uniqueness constraint to each role in a binary fact type means that neither role can include duplicate values. For example, in the fact type Man maries/is married by Woman we want to say that a man maries only one woman and one woman is married by only one man. By applying IUCs to each role in the predicate maries/is married by, we can see that there are no repeated instances in either role position in the sample population table of this fact type (cf. Figure 6.23).

#### 6.4.2.3 Many-to-many

The many-to-many pattern is the weakest uniqueness constraint. In case IUCs span the both roles of a binary predicate, the two object types are in a many-to-many relationship. By default, a binary relationship is many-to-many without condition. Revisit the fact type Department employs/works for Employees, we see



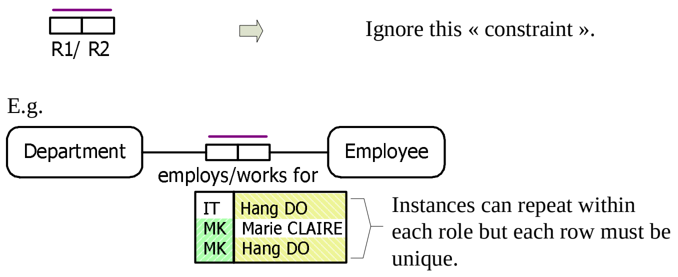
R1/R2
Apply IUC1-rule to R1 and to R2
E.g.
Man
Woman
marries/is married by
Lee Smith Adams
Do White Claire
Each role must include a unique value

```
<owl:Class rdf:about="#Man">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#married"/>
      <owl:maxCardinality
        rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

```
<owl:Class rdf:about="#Woman">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#is_married_by"/>
      <owl:maxCardinality
        rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Figure 6.23: IUC2-rule - Modeling pattern one-to-one of IUCs in OWL

that the predicate employs/works for satisfies this constraint. Hence such "constraints" are simply ignored (cf. Figure 6.24).

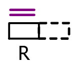


R1/ R2
Ignore this « constraint ».
E.g.
Department
Employee
employs/works for
IT Hang DO
MK Marie CLAIRE
MK Hang DO
Instances can repeat within each role but each row must be unique.

Figure 6.24: IUC3-rule - Modeling the many-to-many pattern of IUCs in OWL

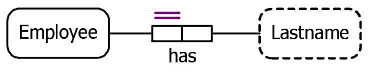
#### 6.4.2.4 Preferred IUC

A *preferred* IUC is indicated by a double line. The notion of preferred uniqueness is conceptual, corresponding to a business decision to prefer a particular identification scheme. Therefore, we use rdfs:comment to translate this remark into an annotation in OWL (cf. Figure 6.25).



i) Apply IUC1-rule to R
ii) Apply <rdfs:comment> annotation to R

E.g.



```
<owl:ObjectProperty rdf:about="#Employee">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#has"/>
      <owl:maxCardinality
         rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:maxCardinality>
      <rdfs:comment> This constraint is preferred
      </rdfs:comment>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:ObjectProperty>
```

Figure 6.25: PIUC-rule - Modeling the preferred IUCs in OWL

### 6.4.3 Frequency Constraints

Internal uniqueness constraints are a special case of *frequency constraints* (FCs). An FC of $n$ (where $n$ is an integer) on a role or a role sequence means any given instantiation of the role (sequence) occurs *exactly* $n$ times (in that relation). Note that when $n = 1$, this constraint is reduced to IUC many-to-one. An FC of $\geq n$ or of $\leq n$ on a role (sequence) means that each instantiation of the role (sequence) occurs at least or at most $n$ times. An FC of $n..m$ on a role (sequence) means that each instantiation of the role (sequence) occurs at least $n$ and at most $m$ times. By allowing that $n = m$, an FC of $n$ may be defined as an FC of $n..m$. By allowing that $n = 0$, an FC of $\leq m$ may be defined as an FC of $n..m$. By allowing that $m$ is a very large number (e.g. 99999), an FC of $\geq n$ may be defined as an FC of $n..m$. So the FC of $n$, of $\leq n$ and of $\geq n$ are special cases of the FC of $n..m$.

For example, to the fact type Department employs Employee, one would apply a frequency constraint to the role played by Department to make sure the required number of employees in each department. If each Department must always employ exactly five employees, place a frequency constraint of 5 on the role played by Department (cf. Figure 6.26(a)). If each Department must employ at least two employees, the constraint is depicted as in Figure 6.26(b). If each Department must employ at most twenty employees, the constraint is depicted as in Figure 6.26(c). If each Department must employ at least five but no more than twenty employees, place a frequency constraint with a minimum of 5 and a maximum of 20 on the role played by Department (cf. Figure 6.26(d)).

Note that we translate role in ORM into property in OWL. Hence, the meaning of FCs shown above can be expressed by cardinality constraints in OWL. OWL pro-

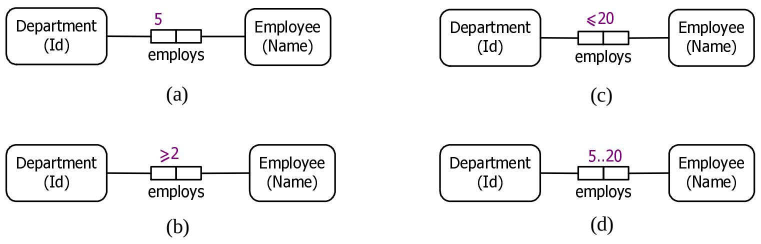
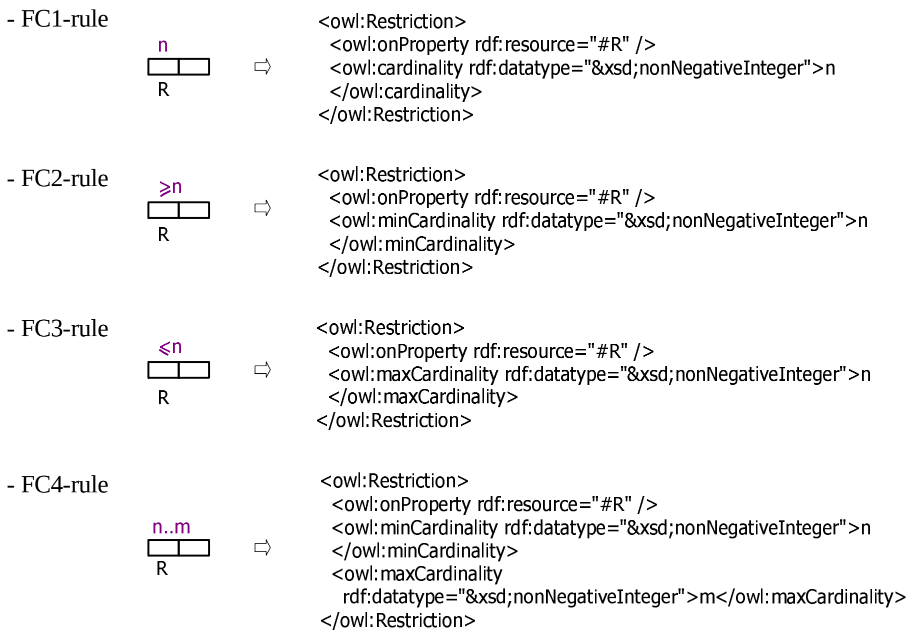


Figure 6.26: Example of using FCs in ORM

vides three constructors for restricting the cardinality of properties locally within a class context. The owl:maxCardinality, owl:minCardinality, and owl:cardinality describe at most, at least, and exactly the number of instances of a class for the concerned property respectively. Therefore, we will use the pair of constructors owl:minCardinality and owl:maxCardinality to translate the FC of $n..m$. owl:minCardinality, owl:maxCardinality, and owl:Cardinality are used to translate FCs of $\geq n$, FCs of $\leq n$, and FCs of $n$ respectively. The detailed translation is given in Figure 6.27. Note that in Chapter 4, we have not presented the constructor owl:cardinality because actually, this constructor is redundant. That is it can always be replaced by a pair of matching owl:minCardinality and owl:maxCardinality constraints with the same value. However, for a convenient shorthand for users, we apply this constructor here instead of the couple of owl:minCardinality and owl:maxCardinality.



- FC1-rule

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#R" />
  <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">n
  </owl:cardinality>
</owl:Restriction>
```

- FC2-rule

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#R" />
  <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">n
  </owl:minCardinality>
</owl:Restriction>
```

- FC3-rule

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#R" />
  <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">n
  </owl:maxCardinality>
</owl:Restriction>
```

- FC4-rule

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#R" />
  <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">n
  </owl:minCardinality>
  <owl:maxCardinality
    rdf:datatype="&xsd;nonNegativeInteger">m</owl:maxCardinality>
</owl:Restriction>
```

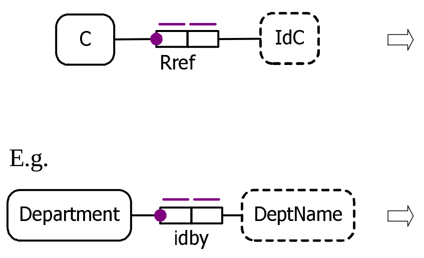
Figure 6.27: Modeling FCs in OWL

### 6.4.4 Simple Reference Schemes

Entities are “real world” objects that are identified by definite descriptions (e.g. the Department with Id “MK”). Such descriptions are defined in terms of ORM as *reference schemes*. In simple cases, a reference scheme indicates that an entity type (e.g. Department) is uniquely identified by a value (e.g. “MK”). We call this scheme *simple reference scheme*.

If an entity type has a simple and preferred reference scheme, this may be abbreviated by a *reference mode* in parenthesis situated next to the entity type (cf. symbol 4 in Figure 6.2). A reference mode is the manner in which the value refers to the entity. It is a convenient way to primarily identify a given instance of an entity. As seen in the previous sections, the entity type Department and its simple reference scheme are defined as Department(Id).

Essentially, simple reference schemes can be explained by adding an attributive role, and by imposing an MC and two IUCs on both the two roles of the predicate. However, the inverse of an attributive role cannot be represented in OWL (cf. Chapter 4). Actually, a simple reference scheme sets up an IC on a class (representing the entity type) by a single data type property (representing attributive role). Hence, we call simple reference schemes *IC patterns* on a single role. Besides, OWL-K supports ICs. Consequently, we use IC assertions in OWL-K to represent this pattern as shown in Figure 6.28.

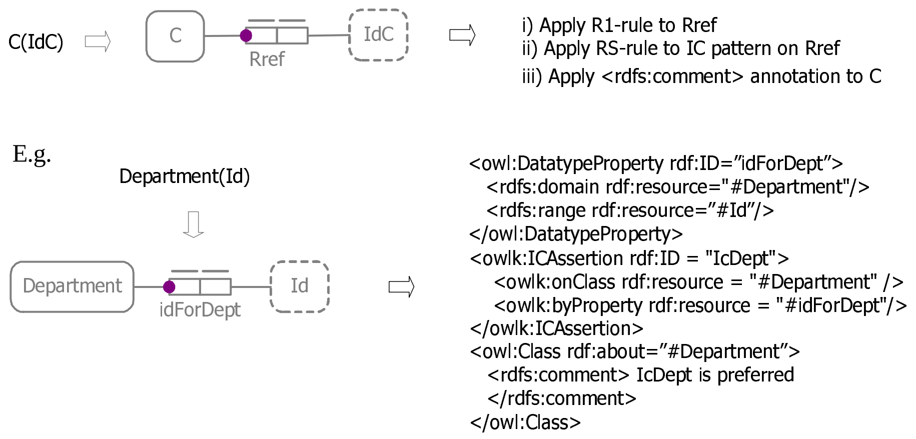


```
<owlk:ICAssertion rdf:ID = "IcC">
   <owlk:onClass rdf:resource = "#C" />
   <owlk:byProperty rdf:resource = "#Rref"/>
</owlk:ICAssertion>
```

```
<owlk:ICAssertion rdf:ID = "IcDeptbyName">
   <owlk:onClass rdf:resource = "#Department" />
   <owlk:byProperty rdf:resource = "#idby"/>
</owlk:ICAssertion>
```

Figure 6.28: RS-rule - Simple reference scheme translation into OWL-K

If a reference mode is used then the simple reference scheme is preferred. Note that in this case, no explicit role name is given to link the entity type to values. Hence, to translate a reference mode we use its equivalence diagram and name a distinct role for the scheme. Similarly to the case of preferred IUC, we use rdfs:comment to translate the preference for the scheme into an annotation in OWL. The detailed translation rule and an example are given in Figure 6.29.



i) Apply R1-rule to Rref
ii) Apply RS-rule to IC pattern on Rref
iii) Apply <rdfs:comment> annotation to C

```
<owl:DatatypeProperty rdf:ID="idForDept">
  <rdfs:domain rdf:resource="#Department"/>
  <rdfs:range rdf:resource="#Id"/>
</owl:DatatypeProperty>
<owlk:ICAssertion rdf:ID = "IcDept">
   <owlk:onClass rdf:resource = "#Department" />
   <owlk:byProperty rdf:resource = "#idForDept"/>
</owlk:ICAssertion>
<owl:Class rdf:about="#Department">
  <rdfs:comment> IcDept is preferred
  </rdfs:comment>
</owl:Class>
```

Figure 6.29: RM-rule - Reference mode translation into OWL-K

## 6.4.5 External Uniqueness Constraints

In case UCs are applied to the roles of many predicates, these constraints are called *external uniqueness constraints* (EUCs). An EUC is denoted by a circle underline which connects to the predicates by dotted lines. Each end of the dotted line connects to a relevant role in a predicate.

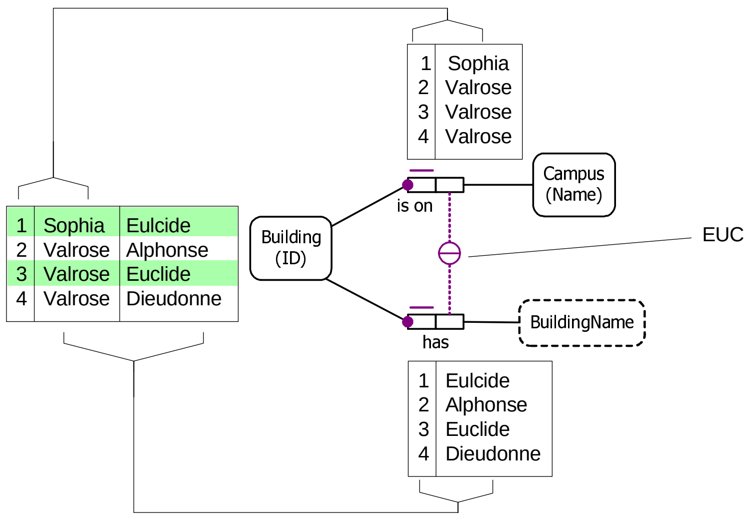


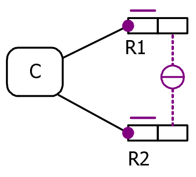
Figure 6.30: The EUC describes no building name duplicated on the same campus

For example, a building name (e.g. Euclide) might occur on different campuses, although it must be unique for each campus. While each building is uniquely identified by its ID, there is still no way of ensuring that no building name is duplicated on the same campus. Figure 6.30 shows how an EUC on roles in two predicates can achieve the desired result.

Essentially, EUCs are used to express unique identification and functional dependency (FD) relationships.

#### 6.4.5.1 EUCs for Unique Identifications

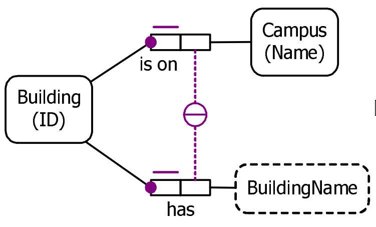
As seen in Figure 6.30, the EUC on the roles played by Campus and BuildingName combined with IUCs and MCs means that an instance of Campus combined with an instance of BuildingName is unique with respect to an instance of a Building. In other words, a building can be uniquely identified by the campus it is on and by its name. Therefore, we call this combination as an identification constraint and translate it, the so-called IC pattern on multiple roles, into an IC assertion in OWL-K. This combination is also given the name *composite reference scheme*. Figure 6.31 gives the translation of IC patterns on the roles of two predicates into OWL-K.



⇨

```
<owlk:ICAssertion rdf:ID = "IcC">
    <owlk:onClass rdf:resource = "#C" />
    <owlk:byProperty rdf:resource = "#R1"/>
    <owlk:byProperty rdf:resource = "#R2"/>
</owlk:ICAssertion>
```

E.g.



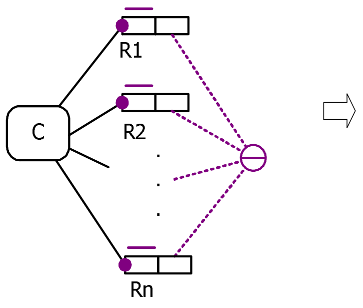
⇨

```
<owlk:ICAssertion rdf:ID = "IcBuilding">
    <owlk:onClass rdf:resource = "#Building" />
    <owlk:byProperty rdf:resource = "#is_on"/>
    <owlk:byProperty rdf:resource = "#has"/>
</owlk:ICAssertion>
```

Figure 6.31: Modeling IC patterns on two roles in OWL-K

The transformation of IC pattern is generalized to the case of $n$ binary predicates as in Figure 6.32.

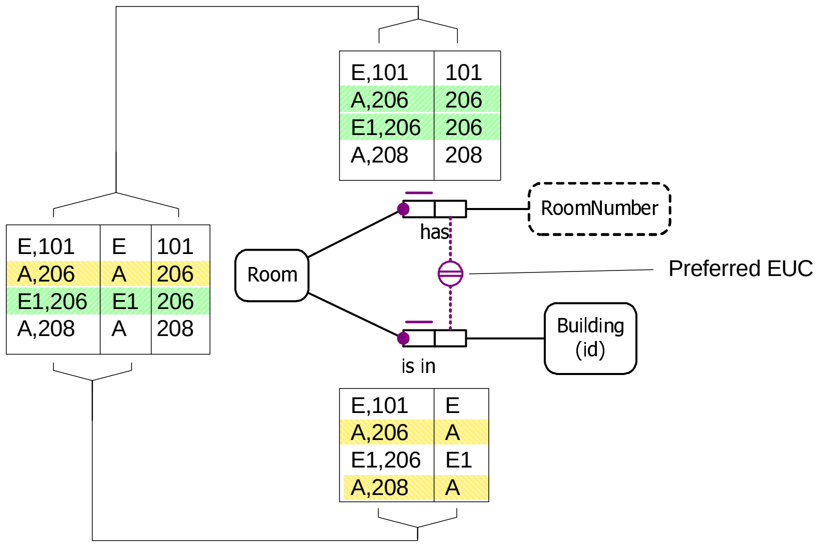
An IC pattern using an EUC is also used to describe a composite type (as mentioned in Section 6.2.2) when chosen as the primary reference scheme of the entity type. In this case, the EUC is denoted by a circled double underline. For example, to look for a room, you go to the building first and then find the room number within that building. Room is, therefore, most naturally described as a composite



```
<owlk:ICAssertion rdf:ID = "IcC">
    <owlk:onClass rdf:resource = "#C" />
    <owlk:byProperty rdf:resource = "#R1"/>
    <owlk:byProperty rdf:resource = "#R2"/>
      ...
    <owlk:byProperty rdf:resource = "#Rn"/>
</owlk:ICAssertion>
```

Figure 6.32: EUC1-rule Modeling IC patterns on $n$ roles in OWL-K

object type that has multiple roles used to reference itself. Figure 6.33 shows how to describe this composite type in ORM using EUC. You could use an identifier, such as a room ID, to identify a room. However, that seems a bit awkward. Moreover, rooms, not room IDs, have other objects such as tables, computers, etc. in them.



| | |
|---|---|
| E,101 | 101 |
| A,206 | 206 |
| E1,206 | 206 |
| A,208 | 208 |

| | | |
|---|---|---|
| E,101 | E | 101 |
| A,206 | A | 206 |
| E1,206 | E1 | 206 |
| A,208 | A | 208 |

| | |
|---|---|
| E,101 | E |
| A,206 | A |
| E1,206 | E1 |
| A,208 | A |

Figure 6.33: The EUC is used to describe composite entity types

A circled double underline denotes that the constraint provides the preferred identification scheme. Similarly to the IUC case, we use rdfs:comment to translate this remark into an annotation in OWL-K (cf. Figure 6.34).

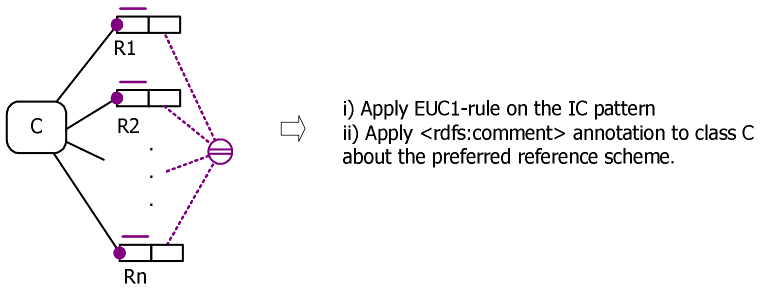
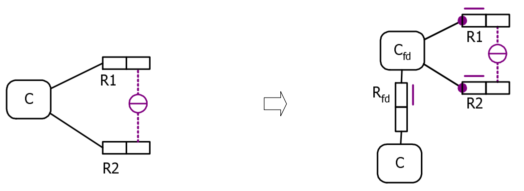

C
R1
R2
Rn
i) Apply EUC1-rule on the IC pattern
ii) Apply <rdfs:comment> annotation to class C
about the preferred reference scheme.

Figure 6.34: PEUC-rule - Modeling IC patterns using the preferred EUCs in OWL-K

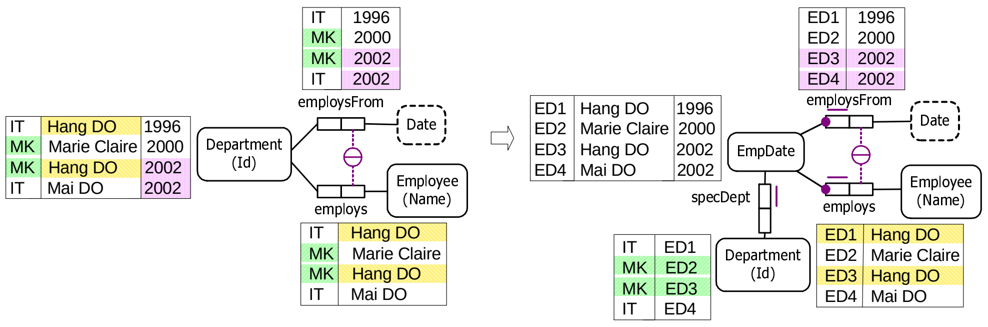
#### 6.4.5.2 EUCs for Functional Dependencies

Except the case described above, the application of EUCs results in FDs. To illustrate FDs represented in ORM, we revisit the example about department in Chapter 2 (section 2.3.4): each department can be specified by an employee and the date from where the department employs him or her. In other words, an entity type Department is functionally dependent on the couple (Date,Employee) through the roles employsFrom and employs respectively. Note that in OWL-K, constructors, except IC assertions, are applied to one role only. In that way, it is not capable of expressing FDs on many roles [23]. However, observe that an FD of one object A on two objects B and C can be described as an FD of A to an object A', which is uniquely identified by the couple (B,C). Hence, we make use of this idea to formalize this kind of EUC. An FD of an object A to an object A' is actually a one-to-many relationship between A and A'. Consequently, this constraint is an IUC in the pattern many-to-one. A', in its turn, is uniquely identified by a pair (B,C). We can describe this by an IC pattern using an EUC on two roles subjecting to B and C respectively. For example, the given EUC on the inverse role of employsFrom and of employs can be replaced by creating an entity EmpDate that specifies the entity type Department and is uniquely identified by the couple (Date,Employee). A new role, so called specDept, is created to set the functional dependency of Department on EmpDate. In Figure 6.35, the ORM schemas on the right are equivalent to those on the left.

Since the one-to-many relationship may be viewed in the inverse direction as the many-to-one pattern of IUC. We employ the latter to describe the former. As a result, FDs can be translated into OWL-K using the translation of many-to-one pattern of IUCs and of an IC pattern using an EUC. Figure 6.36 gives the translation in detail.



E.g.



| | | |
|---|---|---|
| IT | Hang DO | 1996 |
| MK | Marie Claire | 2000 |
| MK | Hang DO | 2002 |
| IT | Mai DO | 2002 |

| | |
|---|---|
| IT | 1996 |
| MK | 2000 |
| MK | 2002 |
| IT | 2002 |

| | |
|---|---|
| IT | Hang DO |
| MK | Marie Claire |
| MK | Hang DO |
| IT | Mai DO |

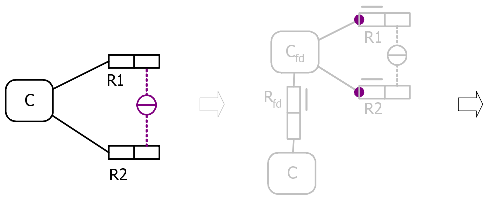
| | | |
|---|---|---|
| ED1 | Hang DO | 1996 |
| ED2 | Marie Claire | 2000 |
| ED3 | Hang DO | 2002 |
| ED4 | Mai DO | 2002 |

| | |
|---|---|
| ED1 | 1996 |
| ED2 | 2000 |
| ED3 | 2002 |
| ED4 | 2002 |

| | |
|---|---|
| IT | ED1 |
| MK | ED2 |
| MK | ED3 |
| IT | ED4 |

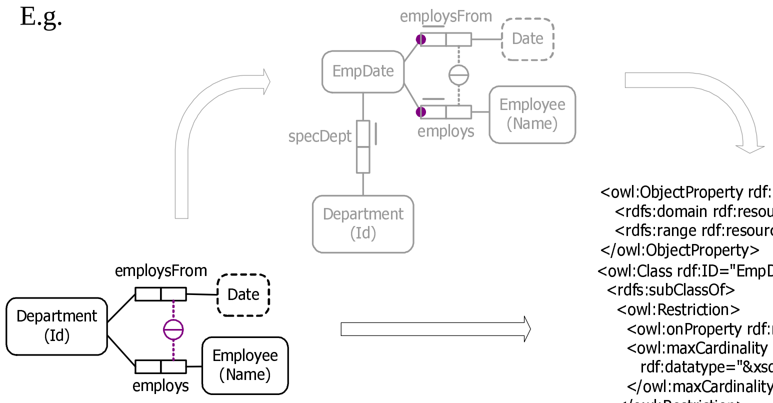
| | |
|---|---|
| ED1 | Hang DO |
| ED2 | Marie Claire |
| ED3 | Hang DO |
| ED4 | Mai DO |

Figure 6.35: ORM schema equivalence



i) Apply R2-rule to $R_{fd}$
ii) Apply IUC1-rule to $R_{fd}$
iii) Apply EUC1-rule to the IC parttern on $C_{fd}$

E.g.



```
<owl:ObjectProperty rdf:ID="specDept">
   <rdfs:domain rdf:resource="#EmpDate"/>
   <rdfs:range rdf:resource="#Department"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="EmpDate">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#specDept"/>
      <owl:maxCardinality
         rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owlk:ICAssertion rdf:ID = "IcEmpDate">
    <owlk:onClass rdf:resource = "#EmpDate" />
    <owlk:byProperty rdf:resource = "#employsFrom"/>
    <owlk:byProperty rdf:resource = "#employs"/>
</owlk:ICAssertion>
```

Figure 6.36: Modeling EUCs on two roles for FDs in OWL-K

The translation of EUCs for FDs is generalized to the case of $n$ binary predicates as in Figure 6.37.

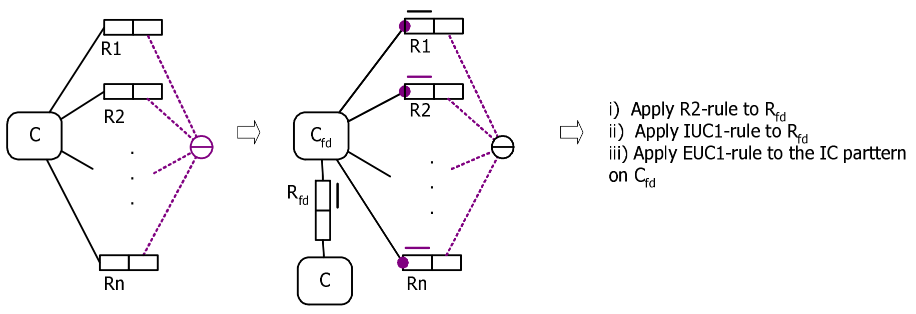

R1
R2
Rn
C
$C_{fd}$
$R_{fd}$
C
R1
R2
Rn
i) Apply R2-rule to $R_{fd}$
ii) Apply IUC1-rule to $R_{fd}$
iii) Apply EUC1-rule to the IC parttern on $C_{fd}$

Figure 6.37: EUC2-rule - Modeling EUCs for FDs in OWL-K

Note that except the combination of constraints that expresses an IC, other constraints are translated separately. It means that we apply the respective rule for each given constraint. For example, in the department example in Chapter 2, we see also the MCs on the roles employsFrom, employs and is_employed_by. These MCs are translated "normally" due to MC2-rule.

### 6.4.6 Nested Object Types

The process of making an object out of a relationship is called *objectification* or also known as *nesting*. The result of objectifying a binary predicate may be viewed as an entity type that is called a nested object type and that has a composite reference scheme whose reference projection bears an equality constraint to the fact type being objectified. To identify a nested object type, a *predicate nesting constraint* is used. This constraint is depicted in an ORM diagram by a soft rectangle around the predicate (called nesting the predicate) in the fact type concerned. A nested object type is given a name. There is nesting of unary predicates and nesting of predicates with non-spanning uniqueness constraints.

Unlike a composite object type, which is identified by multiple roles in two or more fact types, a nested object type is identified by the roles in a single fact type. Actually, nested object types combine their role instances in similar ways to composite object types. Hence, one can think of applying the translation rule for composite object types to nested object types. However, a composite object type plays the roles concerned while a nested object type does not play the roles that identify it. Consequently, this way of translation is not appropriate.

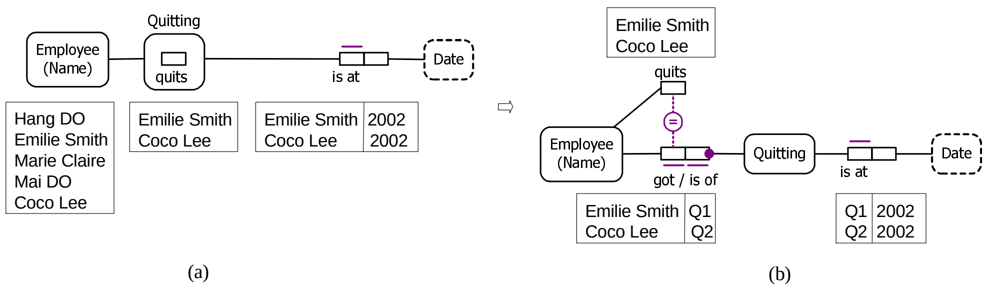


Figure 6.38: An example illustrates the nesting of unary predicates

*Nesting unary predicates.* Figure 6.38(a) shows an example of nesting a unary predicate, which objectifies the fact type Employee quits as Quitting, which is in its turn used as an object type in the fact type Quitting is at Date. Figure 6.38(a) is best understood as an abbreviation of Figure 6.38(b). We call the schema in Figure 6.38(b) is the result of *unraveling* the schema in Figure 6.38(a). Quitting is a normal entity type with linking fact types to Employee and Date. Quitting has a reference scheme because the internal uniqueness constraints on two roles and the mandatory constraint on the link fact type to Employee ensure an injection one-to-one from Quitting to Employee. The equality constraint depicted by a circled "=" indicates that the population of the Employee who quits must be identical to the population of Employee in the fact type Employee got Quitting. Therefore, we try to translate a nested object type in the unary predicate case by translating its explained schema. As shown in Figure 6.38, we need to translate two new roles generated by unraveling the nested object type, the reference scheme and the equality constraint. We have already the rules for translation of roles and of reference schemes. We will deal with the translation of equality constraints in Section 6.4.7.2. Figure 6.39 summarizes our translation rule for nesting unary predicates.

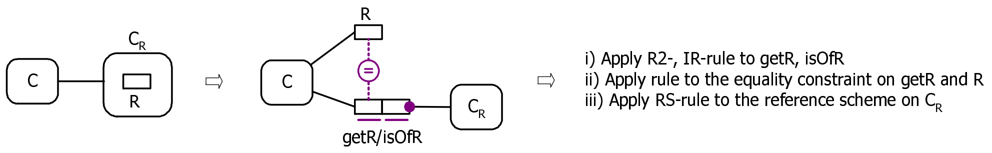
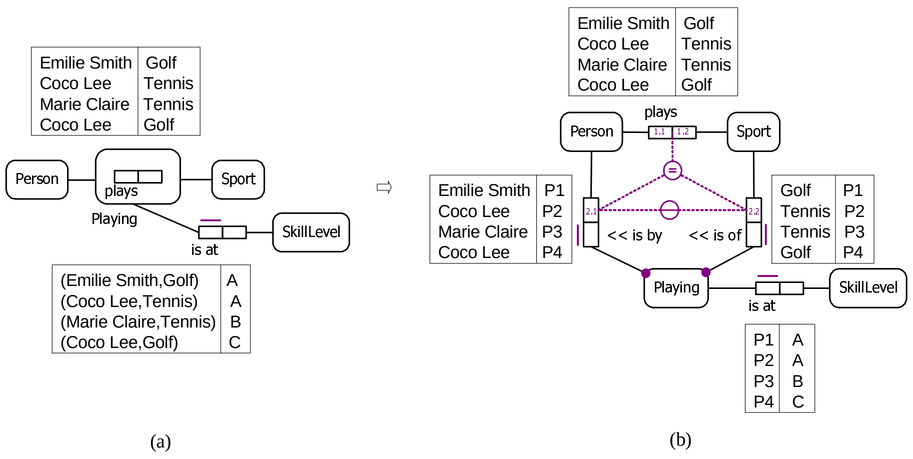

C
$C_R$
R
getR/isOfR
i) Apply R2-, IR-rule to getR, isOfR
ii) Apply rule to the equality constraint on getR and R
iii) Apply RS-rule to the reference scheme on $C_R$

Figure 6.39: NUP-rule - Modeling the nesting of a unary predicate in OWL-K

*Nesting binary predicates with non-spanning uniqueness constraints.* We can see an example of this kind of nesting in Figure 6.40(a). The ORM schema in the figure objectifies the fact type Person plays Sport as Playing, which is then used as an object type in the fact type Playing is at SkillLevel.



| | |
|---|---|
| Emilie Smith | Golf |
| Coco Lee | Tennis |
| Marie Claire | Tennis |
| Coco Lee | Golf |

| | |
|---|---|
| (Emilie Smith,Golf) | A |
| (Coco Lee,Tennis) | A |
| (Marie Claire,Tennis) | B |
| (Coco Lee,Golf) | C |

| | |
|---|---|
| Emilie Smith | Golf |
| Coco Lee | Tennis |
| Marie Claire | Tennis |
| Coco Lee | Golf |

| | |
|---|---|
| Emilie Smith | P1 |
| Coco Lee | P2 |
| Marie Claire | P3 |
| Coco Lee | P4 |

| | |
|---|---|
| Golf | P1 |
| Tennis | P2 |
| Tennis | P3 |
| Golf | P4 |

| | |
|---|---|
| P1 | A |
| P2 | A |
| P3 | B |
| P4 | C |

Figure 6.40: An example illustrates the nesting of unary predicates without IUCs.

Actually, the schema is understood as an abbreviation of Figure 6.40(b). Playing is a normal entity type with linking fact types to Person and Sport. Playing has a composite reference scheme since the external and internal uniqueness, and mandatory constraints on the link fact types ensure an injection one-to-one from Playing to (Person, Sport) pairs. The annotation (1.1, 1.2) indicates a role projection formed by projecting respectively on the left and right roles of the fact type Person plays Sport. The annotation (2.1, 2.2) indicates the reference projection for Playing that is formed by projecting respectively on the link roles played by Person and Sport. Role sequence annotations visually disambiguate those rare cases where the role sequences are otherwise ambiguous. Similarly to the nesting of unary predicates, we translate the nesting of binary predicates by using the translation rules for roles, composite object types, and for equality constraints. The detailed translation rule is given in Figure 6.41.

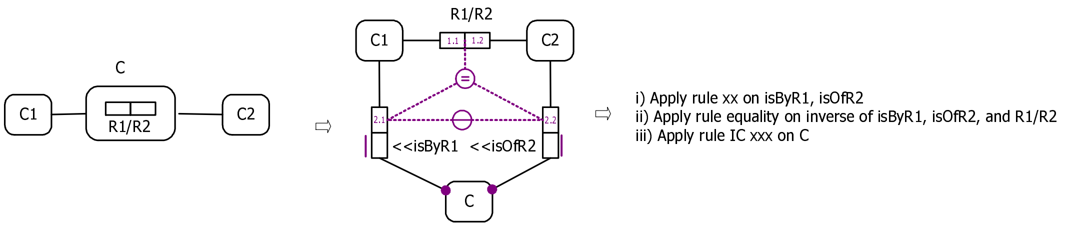

C
C1
R1/R2
C2
⇨
R1/R2
C1
1.1
1.2
C2
=
2.1
2.2
<<isByR1
<<isOfR2
C
⇨
i) Apply rule xx on isByR1, isOfR2
ii) Apply rule equality on inverse of isByR1, isOfR2, and R1/R2
iii) Apply rule IC xxx on C

Figure 6.41: NBP-rule - nesting of binary predicates without IUCs.

For the case of nesting many-to-one (or one-to-many) or one-to-one binary associations, the fact types are usually compound, rather than elementary. Therefore they can be split without information loss into the two simpler fact types. Figure 6.42 gives two examples corresponding to the case of many-to-one (cf. Figure 6.42(a)) and of one-to-one (cf. Figure 6.42(b)) to show that objectifying is not necessary. However, when needed, these kinds of nesting can be also translated into OWL-K using the NBP-rule associated with the relevant IUC rules for the associations.

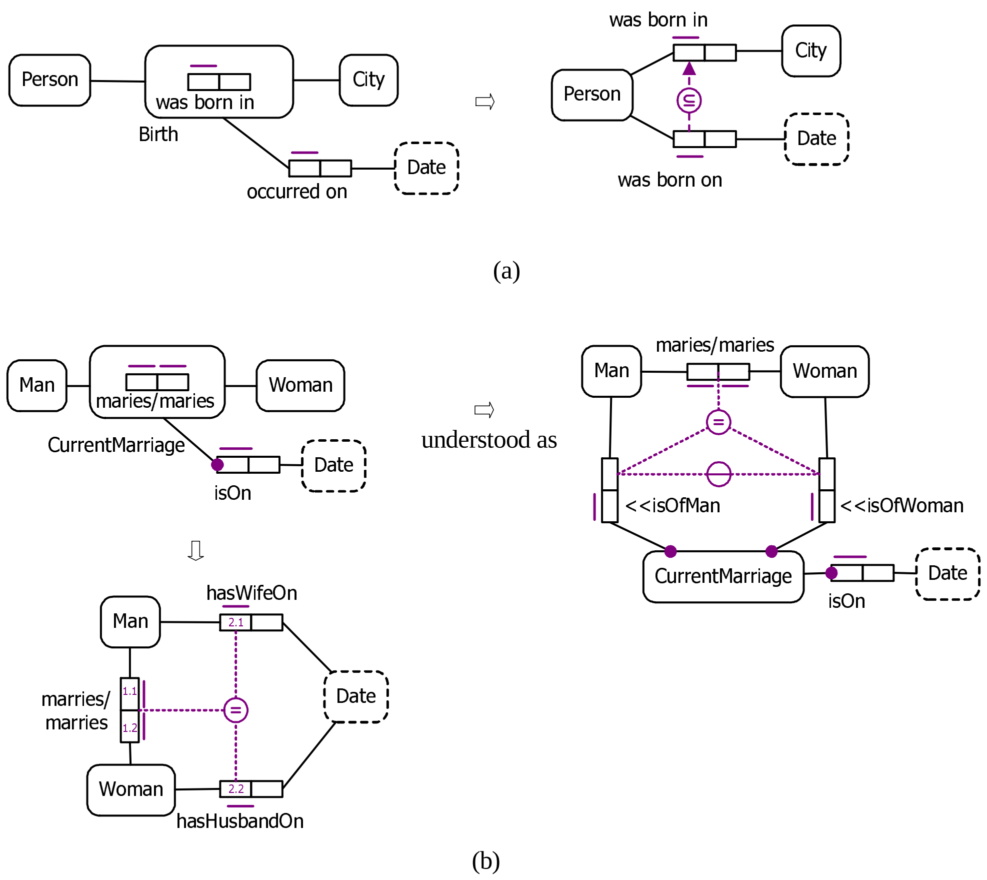


Figure 6.42: Examples showing nested object types split into simpler fact types.

### 6.4.7 Set-comparison Constraints

Subset, equality, and exclusion constraints are denoted by a circle containing $\subseteq, =$, and $\times$ respectively, connected to the associated roles with dashed lines.

### 6.4.7.1 Subset constraints

The subset constraint (SC) restricts the way the population of one role or role sequence relates to the population of another role or role sequence. A subset constraint shows that each instance in the population of one role or role sequence must be included in another. For example, to track the start date and the end date of project, we have a constraint that no project can have an end date unless it has a start date (cf. Figure 6.43).

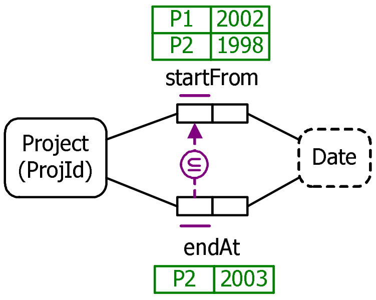
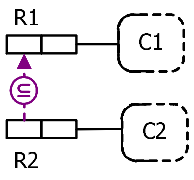


Figure 6.43: Subset constraint describing projects having no end date without a start date.

Figure 6.43 shows a subset constraint between two single roles in two separate fact types, which says that any instances of Project in the lower fact type must also occur in the upper fact type. The arrow should point from the subset role to the superset role. The population of the subset role is a subset of the population of the superset role. It means that you cannot record an end Date for a given Project unless you also record a start Date.

The population of a role describes a class of all individuals for which there exists at least one instance of the object type subjected to by the role. The constructor owl:someValuesFrom describes a class of all individuals for which there exists at least one value of the property concerned. The population is, therefore, can be translated into a class in OWL using the value constraint owl:someValuesFrom. The subset relationship between two populations is then translated into the subclass relationship in OWL. The detailed translation for the subset constraint between two single roles is shown in Figure 6.44. Note that here, we show the case for only two binary fact types. Without loss of generality, we can consider that unary fact types are binary ones where the object type subjected to by the concerned role is the universe, i.e., corresponds to the class owl:Thing in OWL. Therefore, the rule SC1 is also applied to unary fact types.

In case a SC is applied to two role sequences each of which contains two roles in the

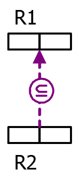


⇨

```
<owl:Restriction>
   <owl:onProperty rdf:resource="#R2"/>
   <owl:someValuesFrom rdf:resource="#C2"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#R1"/>
         <owl:someValuesFrom rdf:resource="#C1"/>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Restriction>
```

Figure 6.44: SC1-rule - Modeling subset constraint between two roles in OWL.

binary predicate, we have a SC between two binary predicates. This constraint restricts that the set of pairs of instances in an entire role sequence must be a subset of the population of the other role sequence. For example, one may use this constraint to express that a department can only fire the employees it has recruited before. This constraint, in other words, describes that one population of a constrained role sequence is subsumed by the other. Therefore, we translate this SC into OWL as a relationship rdfs:subPropertyOf. Figure 6.45 shows the translation rule and the mapping description for the example described above.



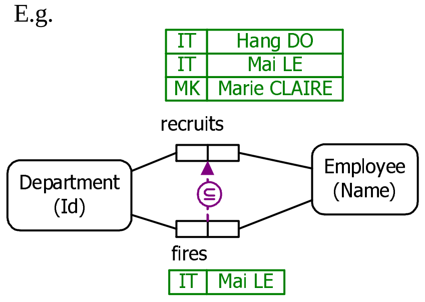
⇨

```
<owl:ObjectProperty rdf:about="#R2">
   <rdfs:subPropertyOf rdf:resource="#R1"/>
</owl:ObjectProperty>
```

or

```
<owl:DatatypeProperty rdf:about="#R2">
   <rdfs:subPropertyOf rdf:resource="#R1"/>
</owl:DatatypeProperty>
```

E.g.



⇨

```
<owl:ObjectProperty rdf:about="#fires">
   <rdfs:subPropertyOf rdf:resource="#recruits"/>
</owl:ObjectProperty>
```

Figure 6.45: SC2-rule - Modeling subset constraint between two binary predicates in OWL.

### 6.4.7.2 Equality constraints

Like the subset constraint, the equality constraint restricts the way the population of one role or role sequence relates to the population of another role or role sequence. An equality constraint sets up a relationship in which the population of the constrained roles or role sequences must be the same. Actually, an equality constraint can be represented by two subset constraints on the same set of constrained roles or role sequences where the subset relationship is set on both two directions (cf. Figure 6.46). Hence, we can apply the rule for subset constraints to the case of equality constraints. Figure 6.46 shows how to apply these rules.

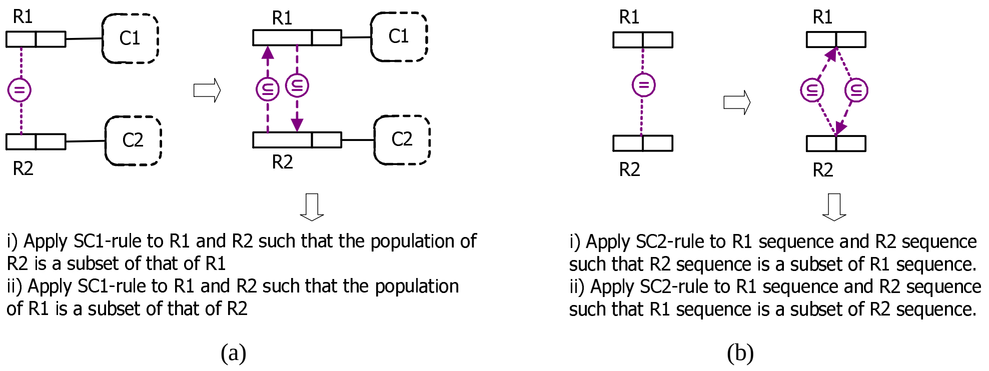

R1
C1
R2
C2
R1
R2
i) Apply SC1-rule to R1 and R2 such that the population of R2 is a subset of that of R1
ii) Apply SC1-rule to R1 and R2 such that the population of R1 is a subset of that of R2
(a)
i) Apply SC2-rule to R1 sequence and R2 sequence such that R2 sequence is a subset of R1 sequence.
ii) Apply SC2-rule to R1 sequence and R2 sequence such that R1 sequence is a subset of R2 sequence.
(b)

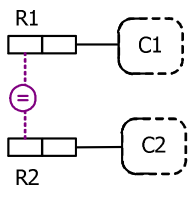
Figure 6.46: Modeling equality constraints using the rules for subset constraints.

Besides, OWL provides the owl:equivalentClass property to describe that the two classes involved have the same set of individuals. Hence, in case an equality constraint is applied to two single roles of two fact types, we replace the two subset relationships (described in Figure 6.46(a)) by the equivalent relationship between the populations of the concerned roles by using the owl:equivalentClass property. For example, to define that a menu is meatetarian if it has meat, one can use the equality constraint between two roles. The rule of translation and its application to this example are shown in Figure 6.47.

Regarding the translation for nested object types (cf. Section 6.4.6), this rule is applied to the case of nesting unary predicates. NUP-rule can thus be rewritten as follows:

i) Apply R2-, IR-rule to getR, isOfR
ii) Apply EQ1-rule to (getR,R)
iii) Apply RS-rule to the reference scheme on $C_R$

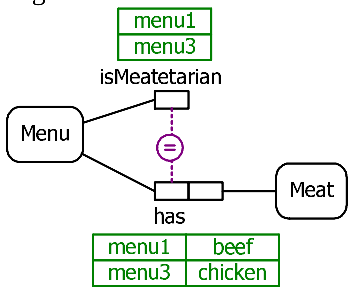
To describe that the set of pairs of instances in an entire role sequence is the same as the population of the other role sequence, we can use the owl:equivalentProperty constructor of OWL, which states that two properties have the same set of instances. Therefore, in case an equality constraint is applied to two binary predicates, we replace the two subset relationships (described in Figure 6.46(b)) by the equivalent relationship between the populations of the concerned roles by using



⇨

```
<owl:Restriction>
   <owl:onProperty rdf:resource="#R2"/>
   <owl:someValuesFrom rdf:resource="#C2"/>
   <owl:equivalentClass>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#R1"/>
         <owl:someValuesFrom rdf:resource="#C1"/>
       </owl:Restriction>
   </owl:equivalentClass>
</owl:Restriction>
```

E.g.

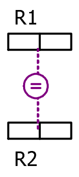


⇨

```
<owl:Restriction>
   <owl:onProperty rdf:resource="#isMeatetarian"/>
   <owl:allValuesFrom rdf:resource="&owl;Thing"/>
   <owl:equivalentClass>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#has"/>
         <owl:someValuesFrom rdf:resource="#Meat"/>
       </owl:Restriction>
   </owl:equivalentClass>
</owl:Restriction>
```

Figure 6.47: EQ1-rule - Modeling equality constraints between two roles in OWL

the owl:equivalentClass property. For example, one may use an equality constraint between two predicates to express that if an employee has a business trip then it will be paid. The rule of translation and its application to this example are shown in Figure 6.48.



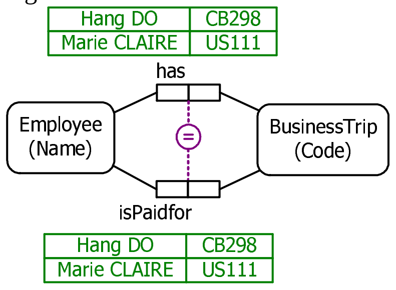
⇨

```
<owl:ObjectProperty rdf:about="#R2">
  <owl:equivalentProperty rdf:resource="#R1"/>
</owl:ObjectProperty>
```

or

```
<owl:DatatypeProperty rdf:about="#R2">
  <owl:equivalentProperty rdf:resource="#R1"/>
</owl:DatatypeProperty>
```

E.g.



⇨

```
<owl:ObjectProperty rdf:about="#isPaidfor">
  <owl:equivalentProperty rdf:resource="#has"/>
</owl:ObjectProperty>
```

Figure 6.48: EQ2-rule - Modeling equality constraints between two predicates in OWL

### 6.4.7.3 Exclusion constraints

An exclusion constraint (EC) is like a DMC except that it sets up a mutually exclusive relationship that is also optional. In other words, each instance in the population of an object type can play only one of the constrained roles but is not required to play any role. For example, if you take a 13 euro menu, you can choose either a dessert or an entre, but not both of them. This expression can be modeled in ORM using an EC as in Figure 6.49(a).

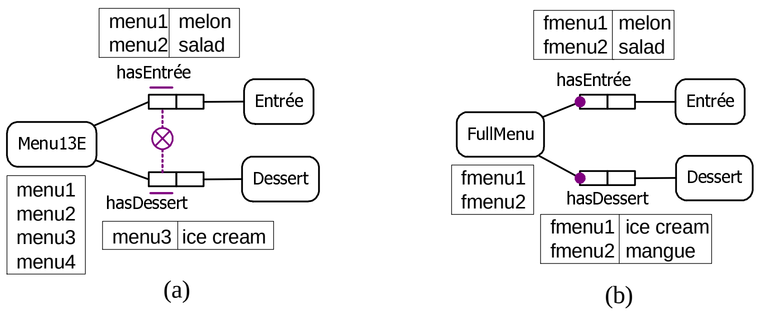
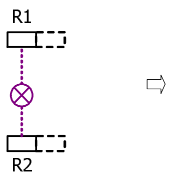
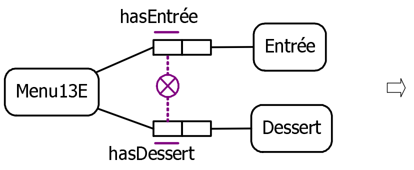


Figure 6.49: The population of Menu13E is *not* the population of FullMenu



```
<owl:complementOf>
    <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
            <Apply MC1-rule or MC2-rule to R1>
            <Apply MC1-rule or MC2-rule to R2>
        </owl:intersectionOf>
    </owl:Class>
</owl:complementOf>
```

E.g.

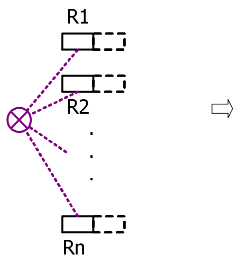


```
<owl:Class rdf:about="#Menu13E">
    <rdfs:subClassOf>
        <owl:Class>
            <owl:complementOf>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasDessert"/>
                            <owl:someValuesFrom rdf:resource="#Dessert"/>
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasEntrée"/>
                            <owl:someValuesFrom rdf:resource="#Entrée"/>
                        </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:complementOf>
        </owl:Class>
    </rdfs:subClassOf>
</owl:Class>
```

Figure 6.50: Modeling ECs on two predicates in OWL

Actually, the population resulting from an EC on the concerned roles is the complement of the population resulting from MCs on all those roles. For example, the expression above can also be restated that the 13 euro menus are *not* the menus consisting of both entre and dessert. This negation can be expressed in OWL by the owl:complementOf property. The object of owl:complementOf is the entity type playing the concerned roles but under the simple MCs. We show the detailed translation for ECs on two predicates in Figure 6.50. For the translation of the above expression, seeing that the menus, so-called full menus, consisting both entre and dessert can be modeled in ORM using MCs (cf. Figure 6.49(b)), we employ full menus as the object of owl:complementOf. The translation is thus given in Figure 6.50.

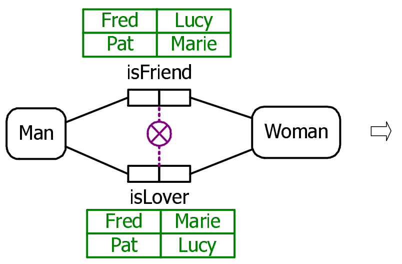
The translation of ECs is generalized to the case of $n$ predicates as in Figure 6.51.



```
<owl:complementOf>
    <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
            <Apply MC1-rule or MC2-rule to R1>
            <Apply MC1-rule or MC2-rule to R2>
                ...
            <Apply MC1-rule or MC2-rule tn Rn>
        </owl:intersectionOf>
    </owl:Class>
</owl:complementOf>
```

Figure 6.51: EC-rule - Modeling ECs on $n$ roles of $n$ predicates in OWL

In an ORM diagram, we may also see an EC between two role sequences. This type of EC enforces a restriction that the instances in an entire role sequence must be different from the instances in the other role sequence.



Need negation constructor for OWL properties!

Figure 6.52: Exclusion constraints for sequences of roles

For example, Figure 6.52 describes that a man is either a friend or a lover of a woman and vice versa. Note that a man or a woman can have both the relationship “is friend” and “is lover”. In this case, the negation of subsumption of one population of a role sequence to the other must be expressed. In OWL terminology, we can say that one property must be *not* subsumed by an other. Unfortunately, so far OWL-K and its underlying DL do not support this kind of expression.

#### 6.4.7.4 General Set-comparison Constraints

In ORM, there are also set comparison constraints which applies to many sequences of one or more roles. A dotted line connects the constraint to roles in the relevant argument for the constraint (cf. Figure 6.53). We call these constraints the general set comparison constraints.

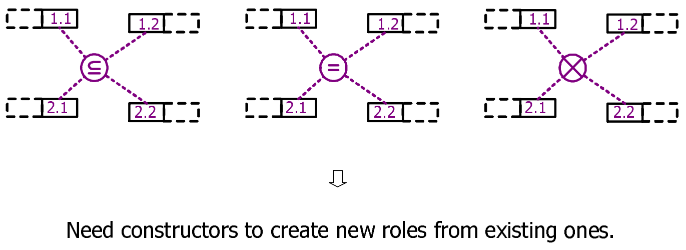


Figure 6.53: General equality constraints in ORM

One example of set-comparison constraints on many sequences is the equality constraint used for explaining nested object types (cf. Figure 6.40). Revisit the example of the nested object type Playing in Section 6.4.6, the equality constraint indicates that the (Person, Sport) pairs in the population of the Person plays Sport fact type must be identical to the population of the (Person, Sport) pairs projected from the Person and Sport roles in the join path Playing is by Person and Playing is of Sport.

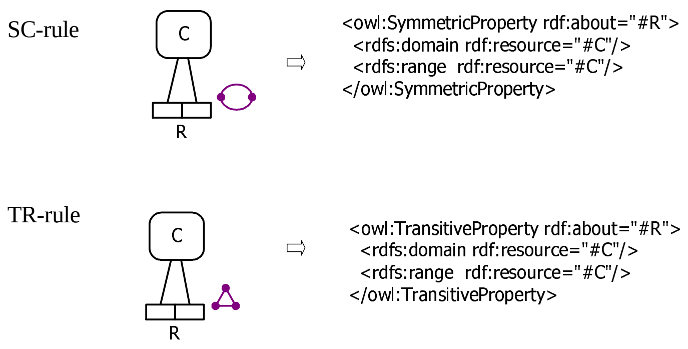
Actually, an equality constraint can be seen as setting the equality of two role sequences that are generated from the populations of the existing roles. However, as presented in Chapter 4, the current standard OWL DL and OWL-K do not support new property creation from existing ones. Hence, the translation of this kind of constraint into the current version of OWL-K is not possible.

### 6.4.8 Ring constraints

A ring constraint applies to a pair of roles played by the same object type in a fact type. Ring constraints are depicted as icons next to the predicates (cf. Figure 6.2). Ring constraints include reflexive and irreflexive (cf. Figure 6.2, symbol 20 top and bottom respectively), symmetric and asymmetric (cf. Figure 6.2, symbol

21 top and bottom respectively), cyclic and acyclic (cf. Figure 6.2, symbol 22 top and bottom respectively), and transitive and intransitive (cf. Figure 6.2, symbol 23 top and bottom respectively). A *reflexive constraint* says that an object can play the both two roles of the predicate at the same time while the irreflexive one does not allow. For example, "Jake cannot be the father of Jake" asserts an irreflexive constraint. A *symmetric constraint* says that if you have a data combination of (A, B), then you have (B, A) while asymmetric constraint does not allow this. For example, if Jake is the father of Joe then Joe cannot be the father of Jake. An *acyclic constraint* says that there are no cycles in a relation. For example, if you have a data combination of (A, B) and (B, C), you cannot have a data combination of (C, A). Thus, if Jake is a father of Joe and Joe is a father of Tom, then Tom cannot be a father of Jake. A *cyclic* constraint allows cycles in a relation. A *transitive constraint* says that if you have a data combination of (A, B) and (B, C), then there exists the data combination of (A, C). Intransitive constraint does not allow this. For example, if Smith is the immediate supervisor of Johnson and Johnson is the immediate supervisor of Davis, then Smith cannot be the immediate supervisor of Davis.

Among these types of ring constraint, only symmetric and transitive constraints can be translated directly into OWL thanks to the built-in OWL class owl:SymmetricProperty and owl:TransitiveProperty. A role under a symmetric or a transitive constraint will be translated into an instance of the class owl:SymmetricProperty or of the class owl:TransitiveProperty respectively. The translation rules and examples are shown in Figure 6.54.



```
<owl:SymmetricProperty rdf:about="#R">
  <rdfs:domain rdf:resource="#C"/>
  <rdfs:range  rdf:resource="#C"/>
</owl:SymmetricProperty>
```

```
<owl:TransitiveProperty rdf:about="#R">
  <rdfs:domain rdf:resource="#C"/>
  <rdfs:range  rdf:resource="#C"/>
</owl:TransitiveProperty>
```

Figure 6.54: Modeling ring constraints in OWL

Other ring constraints require more role constructors. For example, to translate the negation of transitive (i.e. intransitive) or of symmetric (i.e. asymmetric) constraints, we need a way to describe the inequality of the populations of role.

## 6.5 Mapping Algorithm

All the translations analyzed in the previous sections can be summarized in tables 6.2 and 6.3, where DL syntax guarantees the meaning of ORM diagrams, OWL-K abstract syntax relates the semantics to the RDF/XML exchange syntax presented above, and the symbol "-" means that the relevant equality constraint cannot be expressed.

The basic process is to go through the schema, just as an automated system would do, looking for the following descriptions in the schema:

1. Object types (entities and values),
2. Simple reference schemes,
3. Predicates and roles,
4. Constraints,
   (a) Identification constraints on one role,
   (b) Identification constraints on many roles,
   (c) Other constraints.

The detailed algorithm is described as follows.

### 6.5.1 Mapping object types and reference modes

We translate first all the object types in an ORM schema. In ORM, reference modes are described inside the rectangle of the respective entity type. Hence, without influencing other translations we translate reference modes in this step. Moreover, we see that value constraints identify also object types. Consequently, in mapping object types we consider also the value constraints on them.

Given an ORM schema $S_{ORM}$, mapping object types and reference modes defines a procedure $MappingObject(S_{ORM}, S_{OWL-K})$ that maps all object types in $S_{ORM}$, along with their reference modes (if exist) and the value constraints (if exist), to elements of an OWL-K ontology $S_{OWL-K}$ following the relevant rules described in the previous sections (cf. Table 6.2 and Table 6.3). The procedure $MappingObject(S_{ORM}, S_{OWL-K})$ is described in Table 6.4.

Table 6.2: Mapping rules in DL and OWL-K abstract syntax

| | Rule | ORM | DL | OWL-K abstract syntax |
|---|---|---|---|---|
| 1. | VL- | A<br>Range A | A | Datatype URIref A |
| 2. | EN- | C<br>C! | C | Class(C partial) |
| 3. | VC1- | $\{o_1, ..., o_n\}$ | $o_1 \sqcup ... \sqcup o_n$ | oneOf$(o_1 ... o_n)$ |
| 4. | VC2- | $\{v_1, ..., v_n\}$ | $\{v_1, ..., v_n\}$ | oneOf$(v_1 ... v_n)$ |
| 5. | R0- | R in unary predicate played by C | $\geq 1\mathsf{R} \sqsubseteq \mathsf{C}$<br>$\top \sqsubseteq \forall \mathsf{R}.\top$ | ObjectProperty(R domain(C) range(owl:Thing)) |
| 6. | R1- | R played by C subjecting to value type A | $\geq 1\mathsf{R} \sqsubseteq \mathsf{C}$<br>$\top \sqsubseteq \forall \mathsf{R}.\mathsf{A}$ | DatatypeProperty(R domain(C) range(A)) |
| 7. | R2- | R played by $C_1$ subjecting to entity type $C_2$ | $\geq 1\mathsf{R} \sqsubseteq C_1$<br>$\top \sqsubseteq \forall \mathsf{R}.C_2$ | ObjectProperty(R domain($C_1$) range($C_2$)) |
| 8. | IR- | $R_2$ is inverse of $R_1$ | $R_2 \equiv R_1^-$ | ObjectProperty($R_2$ inverseOf($R_1$)) |
| 9. | ST- | $C_1$ is subtype of $C_2$ | $C_1 \sqsubseteq C_2$ | SubClassOf($C_1$ $C_2$) |
| 10. | ME- | $C_1$, ..., $C_n$ mutually exclusive | $C_i \sqsubseteq \neg C_j$<br>$(1 \leq i < j \leq \mathrm{n})$ | DisjointClasses($C_1$... $C_n$) |
| 11. | CE- | $C_1$ ..., $C_n$ exhaust C | $\mathsf{C} \equiv C_1 \sqcap ... \sqcap C_n$ | Classe(C complete $C_1$... $C_n$) |
| 12. | MC1- | MC on R played by C in unary predicate | $\mathsf{C} \sqsubseteq \exists \mathsf{R}.\top$ | Class(C partial restriction (R someValuesFrom($\top$))) |
| 13. | MC2- | MC on R played by $C_1$ subjecting to $C_2$ | $C_1 \sqsubseteq \exists \mathsf{R}.C_2$ | Class($C_1$ partial restriction (R someValuesFrom($C_2$))) |
| 14. | DMC- | DMC on $R_1$, ..., $R_n$ subjecting to $C_1$, ..., $C_n$ respectively | $\exists R_1.C_1 \sqcup ...$<br>$\sqcup \exists R_n.C_n$ | unionOf(restriction($R_1$ someValuesFrom($C_1$))... restriction($R_n$ someValuesFrom($C_n$))) |
| 15. | IUC1- | IUC on R | $\leq 1$ R | restriction(R maxCardinality(1)) |
| 16. | IUC2- | IUC on $R_1/R_2$ | $\leq 1$ $R_1$<br>$\leq 1$ $R_2$ | restriction($R_1$ maxCardinality(1))<br>restriction($R_2$ maxCardinality(1)) |
| 17. | IUC3- | IUC on ($R_1$,$R_2$) | | |
| 18. | PIUC- | preferred IUC on R | $\leq 1$ R | restriction(R maxCardinality(1) annotation(rdfs:comment "..."")) |
| 19. | FC1- | n to R | $\geq$ n R, $\leq$ n R | restriction(R cardinality(n)) |
| 20. | FC2- | $\geq$ n to R | $\geq$ n R | restriction(R minCardinality(n)) |
| 21. | FC3- | $\leq$ n to R | $\leq$ n R | restriction(R maxCardinality(n)) |
| 22. | FC4- | n..m to R | $\geq$ n R, $\leq$ m R | restriction(R minCardinality(n))<br>restriction(R maxCardinality(m)) |
| 23. | RS- | IC pattern on role R for C | R **Idfor** C | ICAssertion(ICAssertionID C R) |
| 24. | RM- | C(Id) | $\geq 1$ $R_{id} \sqsubseteq \mathsf{C}$<br>$\top \sqsubseteq \forall R_{id}.\mathsf{Id}$<br>$R_{id}$ **Idfor** C | DatatypeProperty($R_{id}$ domain(C) range(Id))<br>ICAssertion(ICAssertionID C $R_{id}$)<br>annotation(rdfs:comment "..."")) |

Table 6.3: Mapping rules in DL and OWL-K abstract syntax (cont.)

| | Rule | ORM | DL | OWL-K abstract syntax |
|---|---|---|---|---|
| 25. | EUC1- | IC pattern on ($R_1$,..., $R_n$) for C | $R_1$,..., $R_n$ **Idfor** C | ICAssertion(IdC C $R_1$... $R_n$) |
| 26. | PEUC- | preferred EUC, IC pattern on ($R_1$,..., $R_n$) for C | $R_1$,..., $R_n$ **Idfor** C | ICAssertion(IdC C $R_1$... $R_n$ annotation(rdfs:comment "...")) |
| 27. | EUC2- | EUC as FD for C on $R_1$, $R_2$, ..., $R_n$ | $\geq 1\ R_{fd} \sqsubseteq C_{fd}$ $\top \sqsubseteq \forall\ R_{fd}.C$ $C_{fd} \sqsubseteq \leq 1\ R_{fd}$ | ObjectProperty($R_{fd}$ domain($C_{fd}$) range(C)) SubClassOf($C_{fd}$ restriction(R maxCardinality(1))) |
| | | | $R_1$,..., $R_n$ **Idfor** $C_{fd}$ | ICAssertion(IdC $C_{fd}$ $R_1$...$R_n$) |
| 28. | NUP- | R objectified as $C_R$ | R2-, IR-rule to getR | R2-, IR-rule to getR |
| | | | R2-, IR-rule to isOfR | R2-, IR-rule to isOfR |
| | | | EQ1-rule to (R,getR) | EQ1-rule to (R,getR) |
| | | | RS-rule to $C_R$ | RS-rule to $C_R$ |
| 29. | NBP- | $R_1/R_2$ objectified as $C_R$ | R2-, IR-rule to getR | R2-, IR-rule to getR |
| | | | R2-, IR-rule to isOfR | R2-, IR-rule to isOfR |
| | | | - | - |
| | | | EUC1-rule to $C_R$ | EUC1-rule to $C_R$ |
| 30. | SC1- | subset from $R_2$ subjecting to $C_2$ to $R_1$ subjecting to $C_1$ | $\exists\ R_2.C_2$ $\sqsubseteq$ $\exists\ R_1.C_1$ | SubClassOf(restriction($R_2$ someValuesFrom($C_2$)) restriction($R_1$ someValuesFrom($C_1$))) |
| 31. | SC2- | subset from $R_2$/.. to $R_1$/.. | $R_2 \sqsubseteq R_1$ | SubPropertyOf($R_2$ $R_1$) |
| 32. | EQ1- | equality between | $\exists\ R_2.C_2 \equiv \exists\ R_1.C_1$ | EquivalentClasses( |
| | | $R_2$ subjecting to $C_2$ and | | restriction($R_2$ someValuesFrom($C_2$)) |
| | | $R_1$ subjecting to $C_1$ | | restriction($R_1$ someValuesFrom($C_1$))) |
| 33. | EQ2- | equality between $R_2$/.. and $R_1$/.. | $R_2 \equiv R_1$ | EquivalentProperties($R_2$ $R_1$) |
| 34. | EC- | Exclusion on $R_1$, ..., $R_n$ subjecting to $C_1$, ..., $C_n$ respectively | $\neg(\exists\ R_1.C_1 \sqcap \ldots \sqcap \exists\ R_n.C_n)$ | complementOf(restriction($R_1$ someValuesFrom($C_1$)) ... restriction($R_n$ someValuesFrom($C_n$))) |
| 35. | SC- | symmetric on R | $R \equiv R^-$ | ObjectProperty(R Symmetric) |
| 36. | TR- | transitive on R | Trans(R) | ObjectProperty(R Transitive) |

Table 6.4: Mapping object types and reference modes

```
Procedure $MappingObject(S_{ORM}, S_{OWL-K})$
  For each object type in $S_{ORM}$
    {if object type is entity then
      {if value constraints exists then
         Mapping object type by the VC1-rule
       else Mapping object type by the EN-rule;
       if Reference mode exists then
         Mapping simple reference scheme by the RS-rule
      }
     else if object type is value then
      if enumerated value constraints exists then
         Mapping object type by the VC2-rule
      else Mapping object type by the VL-rule;
    }
End Procedure
```

## 6.5.2 Mapping Predicates and Roles

A role always stands in a predicate. Mapping a role implies, therefore, mapping its predicate, which, in its turn, means mapping all the roles in the predicate and the inverse relation between those roles. Note that a role in a predicate may not be named because in designing an ORM schema, it may not be used. Therefore, the role that does not have a name will not be mapped in this step.

Given an ORM schema $S_{ORM}$, mapping predicates and roles defines a procedure $MappingRole(S_{ORM}, S_{OWL-K})$ that maps all named roles in $S_{ORM}$ to elements of an OWL-K ontology $S_{OWL-K}$ following the relevant rules described in the previous sections (cf. Table 6.2 and Table 6.3). The procedure $MappingRole(S_{ORM}, S_{OWL-K})$ is described in Table 6.5.

## 6.5.3 Mapping Constraints

Some constraints can set up IC patterns. Therefore, in order not to make redundant translation we need to translate these constraints first. Each constraint in an IC pattern will be labeled “translated” after the mapping. Consequently, the mapping constraint will later look for constraints without label “translated” to

Table 6.5: Mapping predicates and roles

```
Procedure $MappingRole(S_{ORM}, S_{OWL-K})$
  For each predicate in $S_{ORM}$
    For each role in the predicate
      if the role is named then
        {if the predicate is unary then
          Mapping the role by the R0-rule;
         if the predicate is binary then
          if the object type subjected by the role is value then
            Mapping the role by the R1-rule
          else
            {Mapping the role by the R2-rule;
             if the other role of the predicate is named then
               Mapping the role as inverse of the other by the IR-rule;
            }
        }
End Procedure
```

translate. However, we see that an IC pattern on multiple roles may contain the constraints in some IC patterns on single role. Hence, an IC pattern can only be translated if not all of its constraints are labeled "translated". Furthermore, there is also a case that all the constraints in an IC pattern on a single role participate in some IC patterns on multiple roles. Therefore, to avoid missing IC patterns on single role (in case all there constraints are already marked "translated"), we translate IC patterns on single role before IC patterns on multiple roles. By this way, we do not have to check the status "translated" of constraint while mapping IC patterns (both on single and multiple roles). Since IC patterns on multiple roles can contain preferred EUCs or not, the algorithm must check both the cases to apply the relevant rule. Other constraints in an ORM schema, which are not marked "translated" can be translated later in any order. Note that the relation subtype-supertype is also a constraint. Hence, we also translate it in this step.

Given an ORM schema $S_{ORM}$, mapping constraints defines a procedure $MappingConstraint(S_{ORM}, S_{OWL-K})$ that maps all constraints in $S_{ORM}$ to elements of an OWL-K ontology $S_{OWL-K}$ following the relevant rules described in the previous sections (cf. Table 6.2 and Table 6.3). The procedure $MappingConstraint(S_{ORM}, S_{OWL-K})$ is described in Table 6.6.

Table 6.6: Mapping constraints

```
**Procedure** MappingConstraint($S_{ORM}, S_{OWL-K}$);
  **For** each set of constraints generating IC pattern on a single role in $S_{ORM}$
    {Mapping the set of constraints using RS-rule;
     Label the set of constraints = "translated";
    }
  **For** each set of constraints generating IC pattern on multiple roles in $S_{ORM}$
    {**If** EUC is preferred **then**
       Mapping the set of constraints using PEUC-rule
     **else** Mapping the set of constraints using EUC1-rule;
     Label the set of constraints = "translated";
    }
  **For** each constraint not translated in $S_{ORM}$
    {Mapping the constraint using the relevant rule in Table 6.2 or 6.3;
     Label the constraint = "translated";
    }
**End Procedure**
```

### 6.5.4 Mapping Procedure

Given an ORM schema $S_{ORM}$, mapping an ORM schema to an OWL-K ontology defines a procedure $ORM2OWLK(S_{ORM}, S_{OWL-K})$ that maps all components in $S_{ORM}$ to elements in an OWL-K ontology $S_{OWL-K}$. The procedure $ORM2OWLK(S_{ORM}, S_{OWL-K})$ is described in Table 6.7.

Table 6.7: Mapping an ORM schema to an OWL-K ontology

```
**Procedure** ORM2OWLK($S_{ORM}, S_{OWL-K}$)
  MappingObject($S_{ORM}, S_{OWL}$);
  MappingRole($S_{ORM}, S_{OWL}$);
  MappingConstraint($S_{ORM}, S_{OWL}$);
**End Procedure**
```

*Algorithm Correctness.* Following the mapping steps, we translate the components of an ORM schema consecutively. Thus, it is obvious that the resulting OWL-K ontology contains the descriptions identical to the ones that are translated separately from ORM diagram components as introduced in the previous sections.

*Algorithm Complexity.* The algorithm reads an ORM schema in a polynomial way. With the loop $For$, it is easy to see that the mapping takes a polynomial time w.r.t the number of object types, roles, and the constraints in the input schema. With regard to space complexity, this algorithm requires a memory to store a finite number of object types, roles associated with its predicates, and of constraints on them. Consequently, the mapping takes a polynomial space w.r.t the number of object types, roles, and the constraints in the input schema.

## 6.6 Conclusion

In this chapter, we have proposed a new formalization of ORM schemas in a web ontology language. The RDF/XML syntax of ORM schemas helps integrating relational sources modeled in ORM into the semantic web environment. Several formalizations of ORM schemas have also been proposed in the literature [123]. They have been proved very useful with respect to establishing a common understanding of the formal meaning of ORM schemas. However, to the best of our knowledge, none of them has the explicit goal of building a framework to integrate information into the Semantic Web environment.

Unlike the previous works that transform database schemas in a particular annotation into those in RDF/XML without understanding the meaning of database schemas (cf. Chapter 3), our work has showed how the meaning of ORM schemas can be captured in the Semantic Web. Particularly, the semantics of ORM diagrams are guaranteed preserved by their OWL-K abstract syntax associated with DL syntax.

Importantly, the mechanism presented above has overcome the shortcomings of the previous works in representing identification constraints (cf. Chapter 3). Our translation can be performed totally automatically and correctly (in the sense of semantic translation) without human interference. Moreover, the mapping algorithm is rather simple in the sense that it is performed in polynomial space w.r.t the input diagram and terminates in polynomial time. Furthermore, our mechanism is capable of capturing functional dependencies, which is a critical constraint as well.

In this chapter, we have also showed that most of constraints in ORM schemas, described through the main symbols for ORM diagramming, can be semantically translated into OWL-K. We have also pointed out that some constraints, i.e., general set-comparison constraints and nested object types, which is a result of general equality constraint application, and ring constraints, cannot be expressed in OWL-K. The reason is that all of these constraints require, in one way or another, role constructors that OWL-K and its underlying DL do not support. Consequently, this fundamental web ontology language for data integration needs to be extended to fully capture the expressivity of ORM. We will talk about this extension in our topics for future research (cf. Chapter 8).

The next chapter will introduce a process integrating relational data sources into the web semantic environment through constructing a mapping tool and a demonstration.

CHAPTER 7

# Implementation Exploit Information from RDB in Semantic Web

In this chapter, we will present how to use the work introduced in the previous chapters to exploit information from relational databases in the Semantic Web. To illustrate the whole process, we give an example of modeling information in ORM (Section 7.1). Section 7.2 will show how the resulting ORM schema can be implemented into a physical relational database. Also, an ORM schema can be retrieved from a physical databases. Section 7.3 present a way to handle ORM diagrams. Consequently, the ORM diagram example will be applied and the result will be then used to generate a Web ontology (Section 7.4). This is achieved by implementing OWL-K language and developing the Orm2OwlK tool. The last section will summarize the work that we have done from the implementation perspective.

## 7.1 Illustration Example

As shown in Chapter 6, ORM diagrams can describe many complicated and high expressive scenarios. In this chapter, we do not expect to cover all the meanings that ORM can express in illustration. Seeing that identification constraints are the most basic and important factor in data modeling, we give an illustration example focusing on mandatory and uniqueness constraints. All the processes presented later in this chapter will be illustrated by the example. Other cases can be applied similarly. The illustration example is given as follows.

**Example 7.1.1.** *Suppose that one would like to describe the information of the students who attended an International meeting of the best students in the world. Each student had a unique name. Each country was coded by country Id and had a country name. Each subject was coded by subject id and had a subject name. Each country had only one student per subject who was elected to represent his/her country in this discipline. Because several students could have the same name, a student was preferred to be identified uniquely by the country he/she presented and the subject at which he/she was the best. For social events, each student could attend many groups, which were coded by group id. Each group had only one topic so that the groups a student attended must have different topics.*

The information in example 7.1.1 is modeled in an ORM diagram (in ORM2 graphical notation [69]) as shown in Figure 7.1.

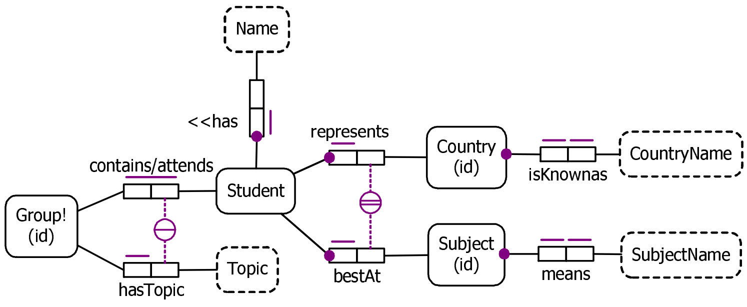


Figure 7.1: ORM schema example

## 7.2 Relational Mapping

For implementation, a conceptual schema is mapped to a logical schema, where the information is grouped into structures supported by the logical data model. The conceptual fact types may be partitioned into sets, with each set mapped to a different table in a relational database schema. Additional non-logical details (e.g. indexes, clustering, column order, final data types, procedures, etc.) may then be specified to complete the internal schema for implementation on the target DBMS.

In particular, after the ORM conceptual model has been specified, mapping takes patterns of facts, objects, roles, and constraints and creates a relational database schema from these patterns. An algorithm is used to group the fact types into normalized tables. If the conceptual fact types are elementary (as they should be), then the mapping is guaranteed to be free of redundancy, since each fact type is grouped into only one table, and fact types which are mapped to the same table all have uniqueness constraints based on the same attribute(s). A simple key may be thought of as a uniqueness constraint spanning exactly one role. A compound key is a uniqueness constraint spanning more than one role. A *compidot* (compositely identified object type) is either a nested object type, or an object type whose primary reference scheme is based on an external uniqueness constraint (e.g. the entity type Room mentioned in the previous chapter). The basic stages in the mapping algorithm are as follows. For more information of this subject, refer to [67].

1. Initially treat each compidot as an atomic "black box" by mentally erasing any predicates used in its identification, and absorb subtypes into their supertype.
2. Map each fact type with a composite key into a separate table, basing the primary key on this key.
3. Group fact types with simple keys attached to a common object type into the same table, basing the primary key on the identifier of this object type.
4. Unravel each mapped compidot into its component attributes.

Conceptual object types are semantic domains. Because current relational systems do not support this feature, domain names are usually omitted. Syntactic domains (data types) may be specified next to the column names if desired.

Roles may be mapped to attribute (column) names. If one role is optional and the other mandatory then the fact type is grouped with the object type on the mandatory side. For example, the Head of Department fact type is grouped into the Department table. A mandatory role is captured by making its corresponding attribute mandatory in its table (not null is assumed by default), by marking as optional all optional roles for the same object type which are mapped to the same table, and by running an equality/subset constraint from those mandatory/optional roles which are mapped to attribute names in another table.

Set-comparison constraints are mapped to corresponding constraints on the attributes to which the relevant roles are mapped. Subtype constraints are mapped to qualifications on optional columns or subset constraints (e.g. foreign key constraints).

Many constraints need to be coded as procedures. For example, if the relation examines is irreflexive, in basic SQL this constraint may be enforced by checking for a null return from:

**select** * **from** examines **where** Examiner = Candidate.

Other refinements to the algorithm have been developed (e.g. mapping of independent object types, certain derived fact type cases, and partially null keys). Conceptual schemas may even be optimized before the relational mapping takes place. The tables produced are in 5th normal form (because the conceptual fact types are elementary).

Both the mapping from the conceptual to the abbreviated relational schema, and the subsequent mapping to the DBMS code can be fully automated[1]. Many ORM conceptual modeling tools, such as NORMA (Neumont ORM Architect) [2], Microsoft Visio for Enterprise Architects (VEA) [40], VisioModeler [103], make mapping easy by automating the entire process. For example, Figure 7.2 shows the relational database schema mapped from the ORM conceptual schema under discussion using the Visio tool.

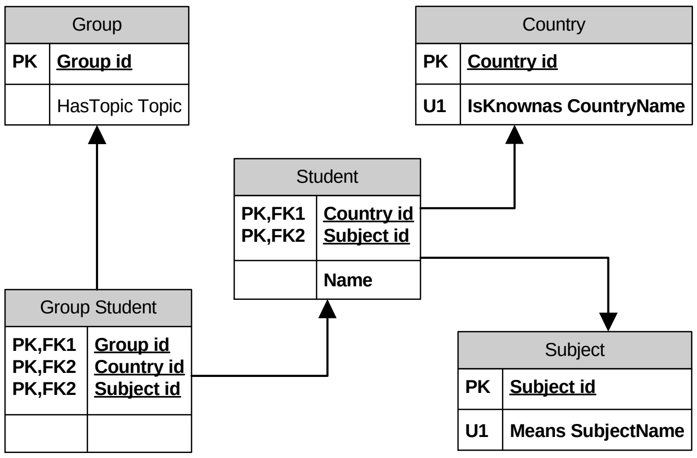


Figure 7.2: The relational DB schema mapped from Figure 7.1

There are five table schemes, with four foreign key connections between them. Each table has its name in the shaded header, with its columns listed below. Primary keys are underlined, marked "PK" and appear in the top compartment

[1]Mapping to the DBMS code depends on the target relational DBMS.

for the columns. Mandatory (not null) columns are in bold. Foreign key columns are marked “FK”n where n is the number of the foreign key with a table. The foreign key connection is depicted as an arrow from the foreign key to its target key. Unique value columns are marked “U”n where n is the number of the unique column with a table.

In this example, the names of the tables and columns are those that are generated automatically by default. In practice they can be renamed and many of the default data types can also be changed. Various configuration options exist for controlling how table and column names are generated.

The DDL script for a selected target DBMS can also be generated. Appendix A presents the DDL script of the ORM schema from Figure 7.1, generated by the VEA tool.

Besides, physical databases could also have their structures reverse engineered to logical database schemas or to ORM schemas (e.g. using the VEA product).

## 7.3 ORM diagrams in XML structure

Diagrams are typically used in off-time mode, i.e. during the design phase. They are not suitable for inter-operability because they depends on specific products. Different products or even one product but different releases often make it difficult to read diagrams by other diagramming software packages as well as other applications. Moreover, some diagram formats, such as Visio format, are not an open format. This means that for exploiting a diagram, the appropriate software package should be installed at site.

Consequently, ORM diagrams should be represented in an open and textual syntax in order to be shared, exchanged, and processed at run-time. Several tools, such as NORMA [2], Orthogonal ToolBox [120], have been developed with this aim. In this thesis, we employ Orthogonal ToolBox which is a free add-on to VEA and can expose ORM schemas as XML documents.

The structure of XML documents describing ORM diagrams are presented in Figure 7.3.

An ORM diagram is described in four sections: Objects, Roles, Facts and Constraints. Each item in the document (i.e. objects, roles, facts, constraints) is identified uniquely by an “id” and has several properties, e.g. “name”, “internal”. Some properties have boolean values. For example, “Independent (T/F)” denotes

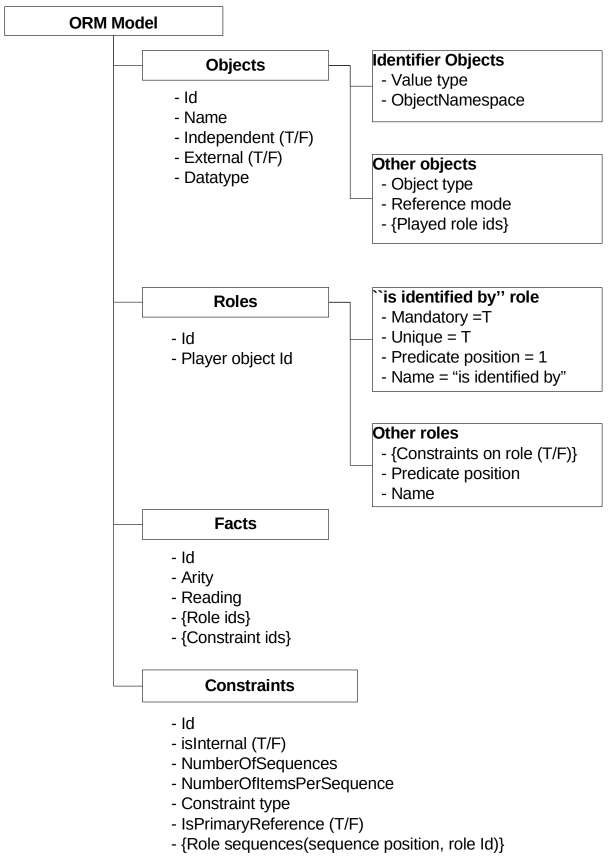


Figure 7.3: The XML schema for ORM diagrams in Orthogonal Toolbox

the property "Independent" of an object can receive the value true (T) if the object type is constrained as independent, or false (F) otherwise.

Objects are classified into two kinds, namely identifier objects that are used as reference mode, and the remains. Besides the common properties of an object, an identifier object is specified as a value type. Moreover, it has a property called "ObjectNamespace" that denotes the object type identified uniquely by it. Other objects are featured by their object type, reference mode, and the set of ids of the roles they play.

A role can also be a "normal" role or a role used for reference mode. All the roles played by identifier objects are named "is identified by" and their properties have specified values, i.e., they are in position 1 of the predicate (Predicate position = 1), under the constraints Mandatory (Mandatory = T) and Uniqueness (Unique =T). Note that those roles do not exist in the respective ORM diagram. However, Orthogonal generates the explanation for reference mode in the XML format of

ORM schemas. This facilitates understanding and extracting the original ORM schemas. Hence, in translating ORM schemas into OWL-K, our implementation can make use of this feature. Other roles are featured by a set of constraints. Each constraint in the set has the value "true' if it is set on the role, and the value "false" otherwise.

Each fact is characterized by its id, its arity, its reading (textual reading based on the predicate), the set of ids of roles in the fact and the set of ids of contraints on the fact. Each constraint can be internal ("isInternal = true") or external (otherwise), preferred ("IsPrimaryReference= true") or not (IsPrimaryReference= false). A constraint has a constraint type (e.g. Mandatory, Uniqueness) and a set of role sequences which consist of sequence positions and ids of the roles it is put on.

The XML document of the ORM diagram in Figure 7.1 can be seen in Appendix B.

## 7.4 Generating Ontologies

### 7.4.1 OWL-K ontologies

*OWL-K built-in vocabulary.* At present, all the modeling primitives of OWL-K extension (w.r.t OWL DL), using the RDF/XML exchange syntax, come from the OWL-K namespace

```
http://dieuthu.free.fr/2007/orm2owl/owlk\#
```

Appendix C contains the corresponding RDF schema for the OWL-K language constructors. This schema provides information about the OWL-K built-in vocabulary extension (w.r.t OWL-DL). Conventionally, OWL(-K) classes have a leading uppercase character, properties a leading lowercase character. Thus, owlk:ICAssertion is a class, and owlk:onClass is a property. The RDF Schema file for OWL-K is not expected to be imported explicitly (i.e., with owl:imports) into an ontology. People who import this schema should expect the resulting ontology to be an OWL Full ontology.

*Content.* An OWL-K ontology starts with a prolog, which is the part of the document that precedes the data. As with most RDF documents, the OWL-K code should be subelements of an rdf:RDF element. This enclosing element generally holds XML namespace and base declarations. Thus, the minimal prolog, containing declarations that identify a document as an OWL-K ontology, should be as in Figure 7.4. An OWL-K ontology consists of any number of class axioms, property axioms, and facts about individuals.

The structure of an OWL-K ontology serialized in RDF/XML is shown in Figure 7.4. Like OWL, an OWL-K ontology in RDF/XML syntax is represented physically in a structured document in plain text with structured elements using tags. The built-in vocabulary is conventionally associated with the namespace name owlk. OWL-K ontologies should not use names from this namespace except for the built-in vocabulary. The definitions of elements in an OWL-K ontology are described between two tags `<rdf:RDF>` and `</rdf:RDF>`. As file extension, an OWL-K ontology can have either .rdf or .owl.

```
<rdf:RDF
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns:owlk="http://dieuthu.free.fr/2007/orm2owl/owlk#"
>

  <owl:Ontology rdf:about=""/>

  <!-- OWL DL and OWL-K code -->

     ...

</rdf:RDF>
```

Figure 7.4: OWL-K ontology structure

### 7.4.2 Orm2OwlK Tool

The ultimate goal of building the Orm2OwlK tool is to set up a mediator for a data integration system. This system will allow one to integrate relational data sources through their ORM schemas into the Semantic Web, and to exploit the original data semantics for various Web services. Within this scope, the tool is also for publishing and sharing OWL-K ontologies on the World Wide Web.

At the first stage, the goal of the tool is to promote the widespread deployment of OWL-K ontologies as well as of data integration using ORM schemas. Moreover, the design of OWL-K should be widely reviewed by researchers, developers and users. Therefore, we have developed the Orm2OwlK tool as a Web application.

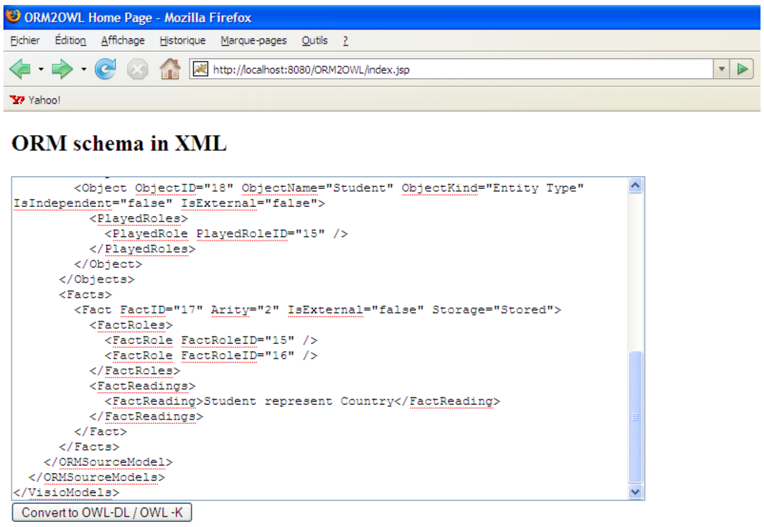
Orm2OwlK is developed using J2EE (Java 2 Platform Enterprise Edition). The latter is a platform-independent, Java-centric environment from Sun for developing, building and deploying Web-based enterprise applications online.

The Java API for XML Processing (JAXP) is an API designed to help writing programs for processing XML documents. JAXP enables applications to parse, transform, validate and query XML documents using an API that is independent of a particular XML processor implementation. The two basic parsing interfaces are the Document Object Model (DOM) parsing interface and the Simple API for XML (SAX) parsing interface. SAX is an event-driven methodology for processing XML. It reads an XML document from beginning to end, and each time it recognizes a syntax construction, it notifies the application that is running it. XML SAX parsers tend to be of low level and may be difficult to program. An important alternative is the DOM parser, defined by the W3C DOM Working Group. It is a set of interfaces for building an object representation, in the form of a tree, of a parsed XML document. Unlike a SAX parser, a DOM parser allows random access to particular pieces of data in an XML document. However, JAXP is compatible with only DOM-Level 1 (DOM-Level 2 is already supported by the most common navigators, e.g. Netscape 6, IE 5). Hence, we employed SAX when it is convenient for implementing our algorithm, and DOM in some cases.

In addition to the parsing interfaces, JAXP provides an XSLT (XML Stylesheet Language for Transformations) interface to make available data and structural transformations on an XML document. For this release of our tool, we employ XSLT as a parser to transform ORM schemas into OWL ontologies.

Orm2OwlK implements the mapping algorithm presented in Chapter 6. It imports ORM schemas in the XML format proposed by Orthogonal (cf. Section 7.3). Because each item in the XML document is identified uniquely by an id, we handle the mapping on only the id numbers of those items. Therefore, the translation does not encounter the problem of name replication, i.e. different items with the same name. In case the role is named “is identified of”, we attach the respective role id to distinguish one from another. Similarly, we name an ICAssertion id for a primary reference scheme the name of the concerned object type attached with its id. For example, Group_51 is the ICAssertion id of the object type Group whose id is 51. We name an ICAssertion id for a reference scheme (not preferred) the name of the concerned object type attached with the id of a role identifying it, e.g. Subject_35. For a new additional element (e.g. in case of functional dependencies), its name will be the combination of two object types (roles) concerned and a unique id if it is an object type (role), e.g. Student_Topic_107. The interface of the tool can be seen in Figure 7.5 and Figure 7.6.

First, an ORM schema in XML format is parsed and loaded into the application by using our input form (cf. Figure 7.5). The schema in XML format is then shown in the window of the tool. In Figure 7.5 we can see a segment of the ORM schema from Figure 7.1 in XML. The button Convert to OWL-DL / OWL -K allows one to translate automatically this ORM schema in XML syntax into an OWL-DL or OWL-K ontology.

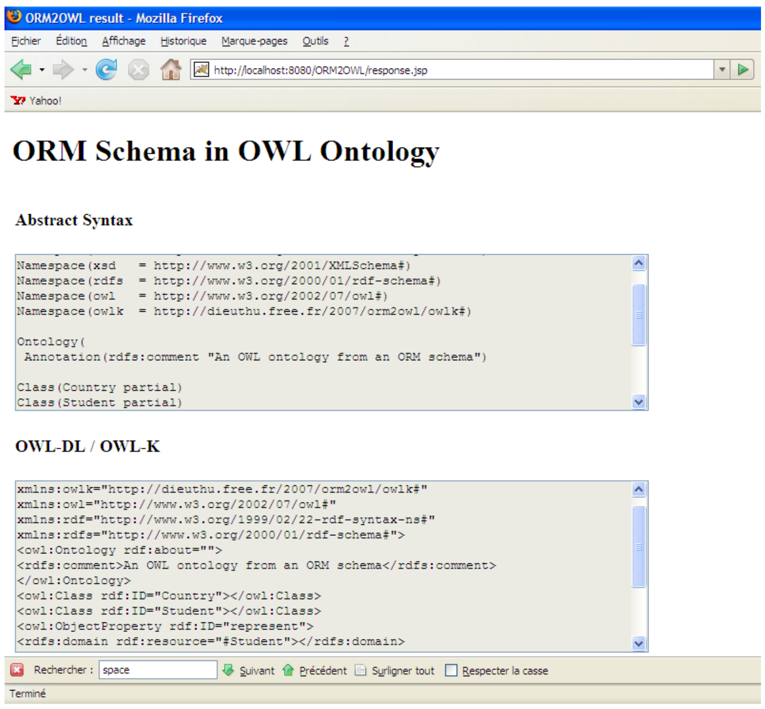

ORM2OWL Home Page - Mozilla Firefox
Fichier Édition Affichage Historique Marque-pages Outils ?
http://localhost:8080/ORM2OWL/index.jsp
Yahoo!
Convert to OWL-DL / OWL -K

ORM schema in XML

```
        <Object ObjectID="18" ObjectName="Student" ObjectKind="Entity Type"
IsIndependent="false" IsExternal="false">
          <PlayedRoles>
            <PlayedRole PlayedRoleID="15" />
          </PlayedRoles>
        </Object>
      </Objects>
      <Facts>
        <Fact FactID="17" Arity="2" IsExternal="false" Storage="Stored">
          <FactRoles>
            <FactRole FactRoleID="15" />
            <FactRole FactRoleID="16" />
          </FactRoles>
          <FactReadings>
            <FactReading>Student represent Country</FactReading>
          </FactReadings>
        </Fact>
      </Facts>
    </ORMSourceModel>
  </ORMSourceModels>
</VisioModels>
```

Figure 7.5: The interface 1 of Orm2OwlK

When one clicks this button, the second interface will be posted (cf. Figure 7.6). This page shows both abstract syntax (the upper window) and RDF/XML syntax (the lower window) of the resulting OWL ontology. The abstract syntax window is for easily understanding the semantics of the respective schema while the RDF/XML syntax window shows the format of the ontology used on the Web. In Figure 7.6, we can see a part of the OWL ontology resulting from the ORM schema given in Figure 7.1. The entire OWL ontology for this schema example is shown in Appendix D.


ORM2OWL result - Mozilla Firefox
Fichier Édition Affichage Historique Marque-pages Outils ?
http://localhost:8080/ORM2OWL/response.jsp
Yahoo!
Rechercher : space
Suivant
Précédent
Surligner tout
Respecter la casse
Terminé

ORM Schema in OWL Ontology

Abstract Syntax

```
Namespace(xsd   = http://www.w3.org/2001/XMLSchema#)
Namespace(rdfs  = http://www.w3.org/2000/01/rdf-schema#)
Namespace(owl   = http://www.w3.org/2002/07/owl#)
Namespace(owlk  = http://dieuthu.free.fr/2007/orm2owl/owlk#)

Ontology(
 Annotation(rdfs:comment "An OWL ontology from an ORM schema")

Class(Country partial)
Class(Student partial)
```

OWL-DL / OWL-K

```
xmlns:owlk="http://dieuthu.free.fr/2007/orm2owl/owlk#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
<owl:Ontology rdf:about="">
<rdfs:comment>An OWL ontology from an ORM schema</rdfs:comment>
</owl:Ontology>
<owl:Class rdf:ID="Country"></owl:Class>
<owl:Class rdf:ID="Student"></owl:Class>
<owl:ObjectProperty rdf:ID="represent">
<rdfs:domain rdf:resource="#Student"></rdfs:domain>
```

Figure 7.6: The interface 2 of Orm2OwlK

## 7.5 Conclusion

In this chapter, we have shown how to integrate relational data sources into the Semantic Web with an illustration example. Throughout the chapter, one can have an overview of the whole process to capture the data semantics from relational data sources. All the steps necessary of the process are also clarified in the chapter. Consequently, we can say that:

- The whole semantic extracting process from relational data sources is semi-automatic. The reason is that in general the repository format in which ORM (or other modeling method's) schemas are stored is proprietary or even "closed" inside the CASE tool's software. To incorporate the functions of these tools into a unified system, which can capture automatically the semantics of the relational sources, the coorperation with tool's suppliers may be needed.

- The semantic capturing process consists of three automatic sub process. The first one is effected by conceptual designing tools (and their reverse engines).

The second one is the syntactic transformation of conceptual schemas into XML format, and is realized by some convert tools (e.g. Orthogonal toolbox). The third one, the most important process, is capturing the semantics of a given schema in XML format and represent them in an OWL ontology, which will be, in its turn, exploited by Web services. This process is done automatically by our Orm2OwlK tool.

- There is no human interference in specifying the semantics of the data source during the whole process. Users are only required to trigger a sub process and provide the input for it. As mentioned above, it is possible to do this work without human getting in the way.

What we have been done in this chapter is only at the first stage. The future work will be discussed later in the last section - Conclusion and Perspectives - of this thesis.

CHAPTER 8

# Conclusion and Perspectives

This thesis concludes with a review of the work presented. We summarize our main results and discuss how they meet the needs of data integration in the SW. Based on this discussion, we list open problems and point out further interesting research for data integration from relational data sources into the SW.

## 8.1 Thesis Overview

This thesis proposes the solutions for the problems that hold back the automatic semantic capturing in data integration from relational data sources into the SW in a decidable way (cf. Chapter 3).

First, it proposes using ORM schemas as a common and comprehensive data model to integrate information from relational data sources. An important advantage of this proposition is the flexibility of data integration system: the changes in data semantics can easily be updated in the integration system without repeating the semantic capturing process for the whole databases. Moreover, conceptual modeling in ORM is far more expressive, clearer and less error prone than ER and UML. Hence, the process can be done automatically using ORM schemas.

Second, it proposes an integration into the SW using the most common and latest standard Web Ontology language OWL for many of its advantages, among which are the world-wide usability and the reasoning capability. To realize the integration, it proposes a decidable extension of OWL DL, namely OWL-K, that supports identification constraints. The latter is crucial in designing databases and hence should be expressible in OWL ontologies.

Integration into the SW is proposed to be carried out on the ontology level by incorporating ORM schemas into OWL(-K) ontologies. Hence, data integration can take advantage of more advanced techniques for ontologies (e.g. ontology mappings) than those for relational database schemas.

Regarding a solution for data integration in a decidable way, i.e., automated reasoning is allowed in the integration system so that knowledge consistency can be verified and data information can be retrieved, the thesis proposes a Tableau algorithm as a decision procedure for $\mathcal{SHOIQK}(\mathbf{D})$, which provides the formal semantics of OWL-K. An important feature of this algorithm is its flexibility. That is, the performance of the algorithm depends on the expressivity of the input ontology. It is comparable with the one for OWL DL.

To realize the automatic semantic capturing process, the thesis proposes a formalization of ORM in OWL-K, which includes a set of mapping rules and an algorithm that uses these rules to generate automatically OWL(-K) ontologies from ORM schemas. The thesis shows that the semantics of ORM schemas captured in OWL(-K) ontologies are preserved thanks to its formalization in the DL $\mathcal{SHOINK}(\mathbf{D})$. This automatic process is illustrated by the Orm2OwlK tool.

To sum up, the thesis has demonstrated that our approach provides a flexible and decidable way to integrate relational data sources into the SW and an automatic process to capture data semantics truthfully.

## 8.2 Main Contributions

This thesis contributes to many research fields. Some results in the thesis have been previously published in [106, 107, 108]. First, throughout the thesis, we have introduced a new approach of data integration from relational data into SW and an implementation at the first stage (Chapter 7). Second, the main contributions to the AI field, in particular to the Description logic theory, have been presented in Chapters 5 and 4, in which we have provided insights in the extension of DLs with identification constraints and concrete domains. The contributions of the thesis to the SW has been exposed in Chapters 4 and 5, in which we have proposed a new decidable web ontology language. Although not being considered as the main objective, our new formalization of a conceptual modeling method presented in Chapter 6 makes an important contribution to conceptual modeling field. In what follows, we describe in detail these contributions.

### 8.2.1 Contributions to the Advances in Data Integration

#### 8.2.1.1 Survey of Data Integration From Relational Databases into the SW

Data integration has been a long-standing challenge for the database community and lately for the SW community. It has received steady attention over the past decades, and has now become a prominent area of database and SW research. In this thesis, we have described recent progress in data integration from relational databases into the SW, and discussed the difficulties underlying the integration process. We have focused in particular on data semantic capture, a topic that is not trivial and has not much success in the recent works. We then identified open research issues. The survey demonstrates that data integration into SW lies at the heart of many database and AI problems, and that addressing it will require solutions that mix database and AI techniques.

#### 8.2.1.2 Foundation for Data Integration from Relational Sources into the SW

The work we have done in this thesis is a decisive step to integrate relational data sources into the SW in an automatic way. It will make existing relational database content, particularly semantics, available for SW applications, such as web agents and services. First, our approach supports systems in accommodating their contents, which are either defined by schemas based on structure or defined by ontologies based on semantics. Hence, databases and SW resources can be integrated together. Relational sources described not only by ORM but also by ER or UML schemas can be integrated because ER and UML schemas can be transformed into ORM schemas. Second, our automatic tool can generate ontologies describing 100% accurate semantics of data sources. As such, data semantic extraction is not laborious and error-prone, and integration systems can be interactive and usable by domain experts who are not necessary computer experts. Third, the semantics of data are reflected truthfully and independently in SW ontologies. As a result, data semantics can be reusable for various purposes and be independent from the run-time characteristics of the underlying sources. Moreover, integration systems can be evolved without breaking the backward compatibility thanks to our choice of ORM. Last but not least, a powerful inference engine based on a Tableau algorithm can be used in design of integration system and in exploitation of information from relational databases in the SW. As far as we know, our approach is the unique that uses an expressive Description logic provided with an inference engine for data integration from relational sources into the Semantic Web.

In particular, the work presented in this thesis will prove useful for applications requiring global schema based architecture as well as P2P-based integration architecture:

*Integration system based on global schema.* For semantic web applications using the same domain, integration system based on a global schema may be preferred. Once data schemas are represented in OWL ontologies, the latter sets up a source ontology layer that holds explicit and formal application-independent knowledge and can be increased independently from web services. Domain ontologies (in the role of global schemas), which set up the service-oriented layer, hold application-dependent (domain) knowledge and are influenced by the intended use of the knowledge and possible requirements. Original query in terms of domain ontology will be translated into a union of queries in terms of source ontologies through inter-ontology relationships or ontology mappings. Then the system will translate the queries into those on the respective data schema using our mapping rules. These queries, in its turn, is translated into SQL queries on databases. The answers can be translated back into those in terms of ontology by using our mapping rules, and then are finally returned to the Web user through ontology mappings between the source and domain ontologies. The process is shown in Figure 8.1(a).

*Integration system based on P2P Architecture.* In a P2P system, there is no existing domain ontology. After we get the OWL ontologies from data models, we can put the OWL ontologies on the web. Suppose some semantic web agents or services wish to access (e.g. query) the data in databases, they use some semantic web query language to send queries to the databases through the OWL ontology at a point. Original queries will be transferred to other points in terms of target ontologies through frames of reference for their queries. Through our mapping rules between OWL ontologies and ORM schemas, queries on ontologies can be transferred to queries on schemas, which are, in their turn, translated into SQL queries on databases. The answers can then be transferred back to the ontology layer to reply the user on the Web by using our mapping rules between ontology layer and schema layer. The process is illustrated in Figure 8.1(b).

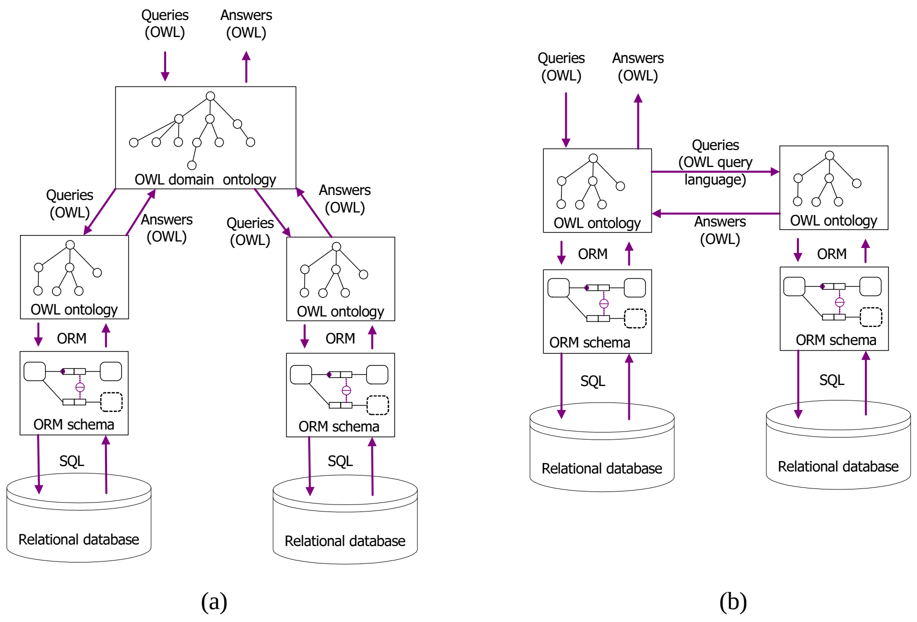


Figure 8.1: Integrating relational databases into the Semantic Web using OWL-K and ORM schemas

### 8.2.2 Contributions to the Advances in the Theory of Description Logics

To deal with reasoning for data integration systems, we provide a flexible DL reasoning procedure which supports concrete datatypes and identification constraints.

- *Concrete domains for $\mathcal{SHOIQ}$.* Integrating Universal concrete domains into $\mathcal{SHOQ}$ has been introduced a long time ago [83]. Nevertheless, applying this feature to $\mathcal{SHOIQ}$ has not been presented in the literature until our work is done. Essentially, integrating concrete domains into $\mathcal{SHOIQ}$ is not a trivial task because this DL has infinite model property. In this thesis, we have introduced a solution to deal with it. Furthermore, we have also shown that this integration works well with another new feature, i.e. identification constraints, by introducing how to reason in presence of concrete domains and ICs, and demonstrated that our algorithm is still decidable with its properties relating to concrete domains.

- *Identification constraints integrated into $\mathcal{SHOIQ}(\mathbf{D})$.* Although DL languages nowadays can provide a very high expressivity, identification constraints are not supported in their nature. In this thesis, we have extended $\mathcal{SHOIQ}(\mathbf{D})$, which is a bit more expressive than the DL underlying the standard SW ontology language OWL DL, to fully capture the semantics of ICs. The extended language is called $\mathcal{SHOIQK}(\mathbf{D})$. We have proposed a practical Tableau algorithm for this new language. The algorithm is proved sound, complete, and terminating. Hence, the extended DL we introduced is decidable. Furthermore, we have designed the algorithm that behaves like "pay as you go" which increases its usability and performance. Our extension has met user requirements because all the things in the real world, in some way, need to be identified and hence this kind of constraint is very important in knowledge representation.

### 8.2.3 Contribution to Web Ontology Language Development

Identification constraints have always been high on the list of requests from users in the SW environment. Despite many researches in theory, this kind of constraints, however, has not been available in Web ontology languages yet. The work in this thesis has met this requirement by introducing a decidable extension of OWL DL, namely OWL-K. OWL-K satisfies all the requirements for a SW ontology language to be applied not only in data integration but also in other Web applications required by ontology users and developers:

1. Provides the "real" ICs;
2. Supports ICs on specific classes, or so called non-global ICs;
3. Provides ICs for data type properties, object properties, and sets of these two kinds of properties;
4. Supports both kinds of constraints corresponding to simple and compound keys in DB schemas;
5. Layers on top of OWL DL;
6. Has formal semantics; and
7. Is a decidable language.

Representation of ICs in the Semantic Web plays an important role. It assists in the designing phase of ontologies and provides the capacity to handle the consistency of KBs. It also provides the capacity to resolve the problem of interoperability and semantic integration of heterogeneous data sources, particularly of relational data sources in the Web environment.

### 8.2.4 Contribution to the Advances in the Theory of Conceptual Modeling

Several formalizations of ORM schemas have been proposed in the literature [65, 123, 90]. They have been proved very useful with respect to establishing a common understanding of the formal meaning of ORM schemas. However, the logics used in their works are not decidable.

In this thesis, we have shown that the DL we use to formalize ORM schemas is decidable. Consequently, our formalization can be considered as an approach that not only satisfies the above objective but also allows automated reasoning on ORM schemas (e.g., detecting constraint conflicts or implications).

## 8.3 Open Problems and Future Work

Although the data semantics can be captured in the SW by our solution, we do not mean that the work we have introduced in this thesis solves all the problems of data integration in the SW. We are just at the beginning of a new and exciting research field and there is still much work to do. In this section, we point out open problems and suggest future research directions.

### 8.3.1 Concrete Domains for $\mathcal{SHOIQK}$

In this thesis, we do not address the concrete reasoner and assume that $\mathcal{SHOIQK}(\mathbf{D})$ has an algorithm satisfying our tableau algorithm requirement. The latter is datatype sets for $\mathcal{SHOIQK}(\mathbf{D})$ are $\mathcal{QIK}$-conforming. That is, in a model all values that share the same datatype associated with an infinite domain are used only once. More research should be needed to remove this constraint while still guaranteeing the termination of the algorithm.

One may say that in reality an infinite datatype domain can be considered as a finite datatype domain whose number of data values is large enough and vice versa. Moreover, ABox is assumed to be an open world. Therefore, without loss of generality, for a practical algorithm, he can define that a *non-determining* individual is the one who has certain specific relation with $\perp$ (or unknown individuals/values). The unraveling process will terminate when it reaches a non-determining node. Consequently, he can define that if in a unraveled chain there exists an identified node all the identifying nodes of which are equals to some others existing nodes, then such an identified node is non-determining. That is, all the identifying nodes will be considered as representations of unknown values and will be removed from the chain. The edges connecting the identified node to these nodes become representations of "relations with $\perp$". As a result, the $\mathcal{QK}$-conformity assumption is still valid for the algorithm and he can remove the $\mathcal{QIK}$-conformity assumption. In this case, with the same argument (or proof), the algorithm is terminating because unraveled chains are finite and no infinite datatype domain is employed in TBoxes as well as in KBoxes.

However, this approach have to solve the problems raised with real finite domains, or else it may give a resulting model by mistake.

### 8.3.2 Complexity of Reasoning Procedure

We have shown that if there exists an algorithm for $\mathcal{QIK}$-conformity in NP, our Tableau algorithm is in 2-NExpTime-complete. However, the upper bound may be lower and we have not investigated this problem in the thesis yet. If the algorithm is proved to have a lower complexity, it will be obviously more attractive to application. Thus, this problem should be solved in the future.

### 8.3.3 Implementation, Evaluation, and Optimization of Reasoning Procedure

To realize our approach in data integration systems, which include exploitation of data semantics and hence reasoning on them, an implementation of the reasoning procedure for $\mathcal{SHOIQK}(\mathbf{D})$ is needed. The fact that the sub language of $\mathcal{SHOIQK}(\mathbf{D})$ (i.e. $\mathcal{SHOIQ}$) has a difficult entailment problem (NExpTime-complete) but still practical is essentially thanks to much work in optimization. The latter is not a trivial task that requires much more research (e.g. [76], [133]). Consequently, we have left the implementation of our Tableau algorithm described in this thesis to the future work, which includes also an evaluation and the study of optimization techniques to improve the performance of the algorithm.

### 8.3.4 IC Representation in DLs and Web Ontology Languages

For simplicity, we allow only simple roles in IC descriptions. Therefore, we should think of a mechanism to inform users about that. Besides, we expect to investigate the presence of transitive roles in IC descriptions and reasoning decidability in this case.

We defined ICs without qualified restrictions. That is, an individual is identified uniquely by a set of any other individuals (in the universe) related to the individual by roles in the appropriate IC description. Note that in $\mathcal{SHOIQK}(\mathbf{D})$, one can specify that an individual may be related to individuals of different concepts by the same role $R$. For example, a TBox could have two axioms $GoodProd \sqsubseteq \geq 3\, hasIngredient.Fruit$ and $GoodProd \sqsubseteq \leq 1\, hasIngredient.Fat$. Therefore, one question to be considered is whether we can employ qualified restrictions in ICs, for example $R.D, S.A$ **Idfor** $C$ ?

Note that this question is worth our attention because many ontology design tools support already qualified cardinality restrictions (cf. Chapter 5). Furthermore, OWL 1.1, an extension of OWL DL being submitted to W3C as a standard Web Ontology language, supports also this feature.

### 8.3.5 ORM Schema Representability in Web Ontology Languages

*Datatype limitations.* In Chapter 7, we see that data types resulting from value type mapping are not shown in the resulting OWL(-K) ontology. Actually, even supporting RDF(S) specification of data types, OWL (including OWL DL and OWL-K as well) does not have a mechanism to access customized data types. The reason is that the XML schema type system does not provide a standard way to access the URI references of user-derived (customized) data types. This drawback makes XML Schema user-derived data types not accessible by RDF(S) and OWL.

*Role description limitations.* As seen in Chapter 6, OWL DL and OWL-K cannot describe general set-comparison constraints and ring constraints in terms of ORM. These constraints require, actually, some role constructors that OWL DL, and thus OWL-K do not support.

Lately, based on requests of major users of OWL DL, OWL 1.1 has been proposed that deals with these shortcomings [125]. OWL 1.1 allows user-defined datatypes to be defined in an OWL ontology. Besides, OWL 1.1 provides a more powerful way to describe properties such as reflexive, irreflexive, symmetric, antisymmetric, and disjoint properties, and property chain inclusion axioms. The specification of OWL 1.1 is currently being discussed and OWL 1.1 is now a W3C Member Submission.

Therefore, expecting that OWL 1.1 will become standardized in the near future, we plan to apply OWL 1.1 to data integration from relational sources into the Semantic Web. Consequently, the future work should be:

1. Integrating universal concrete domain into the DL underlying OWL 1.1. Even OWL 1.1 supports data types, the given DL language underlying OWL 1.1 does not take concrete domains into account [125]. This is still an open problem to be dealt with in the future.

2. Investigating the expressivity of OWL 1.1 in representing ORM schemas. Because OWL 1.1 is an extended of OWL DL, not of OWL-K, we should first verifying the capability of OWL 1.1 in capturing identification constraints. If this is not possible, more work need to be done such as:

    - Integrating ICs into OWL 1.1 considering IC representation perspectives mentioned above.
    - Developing a practical reasoning procedure for the new extension language considering concrete domain influence.

3. Trying to describe the ORM constraints that cannot be described in OWL-K in OWL 1.1 or its extension with ICs (in case ICs should be integrated). If the expressivity of the Web ontology language is still lower than that of ORM, we should consider a further extension to the Web ontology language while guaranteeing its decidability. Hence, a practical reasoning procedure should also be considered.

4. Modeling ORM schemas in the new Web ontology language. Note that although OWL 1.1 provides higher expressivity than OWL DL, it has also restrictions on new constructors. Hence, the modeling should consider all these conditions.

5. Implementing the modeling and reasoning mechanisms.

### 8.3.6 Further Steps in Building a Data Integration System

#### 8.3.6.1 Orm2OwlK Tool Enhancement

In the next stage of developing the tool, along with the implementation of the data integration system, two scenarios may be proposed:

- Create the tool as a plug-in component in conceptual model design tools. Hence when an enterprize wants to share their databases in the Semantic Web, the conceptual schemas of the required databases will be formalized as Web ontologie(s).
- Create the tool as a component of the mediator in a data integration system. The system will use the mapping rules to access information in databases when necessary. There must have a way to connect among physical databases, their conceptual schemas, and their respective Web ontologies to enable transparent inter-operability.

#### 8.3.6.2 Query Processing

To exploit information from relational sources, our future research in building a data integration system should be query processing which focuses on the automatic translation of queries on SW applications to queries on relational databases. Identification constraints help to capture the complex interrelationships in the domain of interest better. However, they have a deep impact on how certain answers are computed, and hence they must be fully taken into account during query answering. Consequently, a query language supporting IC for OWL-K should be introduced. Then based on the formalization presented in this thesis, the translation will make queries represented in the query language for OWL-K ontologies to queries in a query language for ORM schemas (e.g. Conquer [21]). Because queries on ORM schemas can be converted into queries on relational DBMS (cf. Chapter 2), information in relational DBMS can be retrieved by this way.

APPENDIX A

# DDL Script from ORM Schema

The following DDL script is generated from the ORM schema example in chapter 7 for an Access DBMS, and by the VEA tool.

```
create table ‘Subject‘ (
    ‘Subject id‘ CHAR(10),
    ‘Means SubjectName‘ CHAR(10),
    constraint ‘Subject_PK‘ primary key (‘Subject id‘) );

create table ‘Group contain Student‘ (
    ‘Group id‘ CHAR(10),
    ‘Country id‘ CHAR(10),
    ‘Subject id‘ CHAR(10),
    constraint ‘Group contain Student_PK‘ primary key
    (‘Group id‘, ‘Country id‘, ‘Subject id‘) );

create table ‘Student‘ (
    ‘Country id‘ CHAR(10),
    ‘Subject id‘ CHAR(10),
    ‘Name‘ CHAR(10),
    constraint ‘Student_PK‘ primary key (‘Country id‘, ‘Subject id‘) );

create table ‘Group‘ (
    ‘Group id‘ CHAR(10),
    ‘HasTopic Topic‘ CHAR(10),
    constraint ‘Group_PK‘ primary key (‘Group id‘) );

alter table ‘Subject‘ add constraint ‘Subject_UC1‘ unique (
    ‘Means SubjectName‘);
```

```
alter table ‘Group contain Student‘
    add constraint ‘Student_Group contain Student_FK1‘ foreign key (
        ‘Country id‘,
        ‘Subject id‘)
     references ‘Student‘ (
        ‘Country id‘,
        ‘Subject id‘);

alter table ‘Group contain Student‘
    add constraint ‘Group_Group contain Student_FK1‘ foreign key (
        ‘Group id‘)
     references ‘Group‘ (
        ‘Group id‘);

alter table ‘Student‘
    add constraint ‘Subject_Student_FK1‘ foreign key (
        ‘Subject id‘)
     references ‘Subject‘ (
        ‘Subject id‘);
```

APPENDIX B

# ORM Schema in XML

This appendix shows the content of the XML document converted from the ORM schema example in Chapter 7.

```
<?xml version="1.0" encoding="utf-8"?>

<!--Generated by Orthogonal Toolbox (v1.5.1712.33831)
    on Tuesday, January 15, 2008 at 7:14:00 AM-->
<!--Orthogonal Toolbox is a free utility produced by
    Orthogonal Software Corporation -->
<!--http://www.orthogonalsoftware.com -->

<VisioModels>
  <ORMSourceModels>
    <ORMSourceModel FileName="D:\Thesis\Example\Student1.vsd"
                    FileSavedOn="1/15/2008 6:39:10 AM">
      <Roles>
        <Role RoleID="15" IsMandatory="true" IsUnique="true" PredicatePosition="1"
              RoleReading="represents" RolePlayerObjectID="18" />
        <Role RoleID="29" IsMandatory="false" IsUnique="true" PredicatePosition="2"
              RolePlayerObjectID="31" />
        <Role RoleID="28" IsMandatory="true" IsUnique="true" PredicatePosition="1"
              RoleReading="isKnownas" RolePlayerObjectID="19" />
        <Role RoleID="23" IsMandatory="false" IsUnique="false" PredicatePosition="2"
              RolePlayerObjectID="25" />
        <Role RoleID="22" IsMandatory="true" IsUnique="true" PredicatePosition="1"
              RoleReading="bestAt" RolePlayerObjectID="18" />
        <Role RoleID="16" IsMandatory="false" IsUnique="false"
              PredicatePosition="2" RolePlayerObjectID="19" />
        <Role RoleID="43" IsMandatory="false" IsUnique="false" PredicatePosition="2"
              RolePlayerObjectID="45" />
        <Role RoleID="42" IsMandatory="true" IsUnique="true" PredicatePosition="1"
              RoleReading="has" RolePlayerObjectID="18" />
        <Role RoleID="36" IsMandatory="false" IsUnique="true" PredicatePosition="2"
              RolePlayerObjectID="38" />
        <Role RoleID="35" IsMandatory="true" IsUnique="true" PredicatePosition="1"
              RoleReading="means" RolePlayerObjectID="25" />
  <Role RoleID="63" IsMandatory="true" IsUnique="true" PredicatePosition="1"
        RoleReading="is identified by" RolePlayerObjectID="51" />
  <Role RoleID="56" IsMandatory="false" IsUnique="true" PredicatePosition="2"
        RolePlayerObjectID="54" />
  <Role RoleID="55" IsMandatory="true" IsUnique="true" PredicatePosition="1"
        RoleReading="is identified by" RolePlayerObjectID="19" />
  <Role RoleID="49" IsMandatory="false" IsUnique="false" PredicatePosition="2"
        RoleReading="attends" RolePlayerObjectID="18" />
  <Role RoleID="48" IsMandatory="false" IsUnique="false" PredicatePosition="1"
        RoleReading="contains" RolePlayerObjectID="51" />
  <Role RoleID="79" IsMandatory="false" IsUnique="false" PredicatePosition="2"
        RolePlayerObjectID="81" />
  <Role RoleID="78" IsMandatory="false" IsUnique="true" PredicatePosition="1"
        RoleReading="hasTopic" RolePlayerObjectID="51" />
  <Role RoleID="72" IsMandatory="false" IsUnique="true" PredicatePosition="2"
        RolePlayerObjectID="70" />
  <Role RoleID="71" IsMandatory="true" IsUnique="true" PredicatePosition="1"
        RoleReading="is identified by" RolePlayerObjectID="25" />
  <Role RoleID="64" IsMandatory="false" IsUnique="true" PredicatePosition="2"
        RolePlayerObjectID="62" />
</Roles>
<Objects>
  <Object ObjectID="31" ObjectName="CountryName"
  ObjectKind="Value Type" IsIndependent="false"
  IsExternal="false" ConceptualDatatype="C-Fixed Length(10)"
  PhysicalDatatype="CHAR(10)">
    <PlayedRoles>
      <PlayedRole PlayedRoleID="29" />
    </PlayedRoles>
  </Object>
  <Object ObjectID="25" ObjectName="Subject" ReferenceMode="id"
  ObjectKind="Entity Type" IsIndependent="false" IsExternal="false"
  ConceptualDatatype="C-Fixed Length(10)"
  PhysicalDatatype="CHAR(10)">
    <PlayedRoles>
      <PlayedRole PlayedRoleID="35" />
      <PlayedRole PlayedRoleID="23" />
    </PlayedRoles>
  </Object>
  <Object ObjectID="19" ObjectName="Country" ReferenceMode="id"
  ObjectKind="Entity Type" IsIndependent="false" IsExternal="false"
  ConceptualDatatype="C-Fixed Length(10)"
  PhysicalDatatype="CHAR(10)">
    <PlayedRoles>
      <PlayedRole PlayedRoleID="28" />
      <PlayedRole PlayedRoleID="16" />
    </PlayedRoles>
  </Object>
  <Object ObjectID="18" ObjectName="Student" ObjectKind="Entity Type"
  IsIndependent="false" IsExternal="false">
    <PlayedRoles>
      <PlayedRole PlayedRoleID="49" />
      <PlayedRole PlayedRoleID="42" />
      <PlayedRole PlayedRoleID="22" />
      <PlayedRole PlayedRoleID="15" />
    </PlayedRoles>
  </Object>
  <Object ObjectID="45" ObjectName="Name" ObjectKind="Value Type"
  IsIndependent="false" IsExternal="false"
  ConceptualDatatype="C-Fixed Length(10)" PhysicalDatatype="CHAR(10)">
    <PlayedRoles>
      <PlayedRole PlayedRoleID="43" />
      </PlayedRoles>
    </Object>
    <Object ObjectID="38" ObjectName="SubjectName" ObjectKind="Value Type"
    IsIndependent="false" IsExternal="false"
    ConceptualDatatype="C-Fixed Length(10)" PhysicalDatatype="CHAR(10)">
      <PlayedRoles>
        <PlayedRole PlayedRoleID="36" />
      </PlayedRoles>
    </Object>
    <Object ObjectID="62" ObjectName="Groupid" ObjectKind="Value Type"
    IsIndependent="false" IsExternal="false" ObjectNamespace="Group"
    ConceptualDatatype="C-Fixed Length(10)"
    PhysicalDatatype="CHAR(10)" />
    <Object ObjectID="54" ObjectName="Countryid" ObjectKind="Value Type"
    IsIndependent="false" IsExternal="false" ObjectNamespace="Country"
    ConceptualDatatype="C-Fixed Length(10)"
    PhysicalDatatype="CHAR(10)" />
    <Object ObjectID="51" ObjectName="Group" ReferenceMode="id"
    ObjectKind="Entity Type" IsIndependent="true" IsExternal="false"
    ConceptualDatatype="C-Fixed Length(10)"
    PhysicalDatatype="CHAR(10)">
      <PlayedRoles>
        <PlayedRole PlayedRoleID="78" />
        <PlayedRole PlayedRoleID="48" />
      </PlayedRoles>
    </Object>
    <Object ObjectID="70" ObjectName="Subjectid" ObjectKind="Value Type"
    IsIndependent="false" IsExternal="false" ObjectNamespace="Subject"
    ConceptualDatatype="C-Fixed Length(10)"
    PhysicalDatatype="CHAR(10)" />
    <Object ObjectID="81" ObjectName="Topic" ObjectKind="Value Type"
    IsIndependent="false" IsExternal="false"
    ConceptualDatatype="C-Fixed Length(10)" PhysicalDatatype="CHAR(10)">
      <PlayedRoles>
        <PlayedRole PlayedRoleID="79" />
      </PlayedRoles>
    </Object>
  </Objects>
  <Facts>
    <Fact FactID="30" Arity="2" IsExternal="false" Storage="Stored">
      <FactRoles>
        <FactRole FactRoleID="28" />
        <FactRole FactRoleID="29" />
      </FactRoles>
      <FactReadings>
        <FactReading>Country isKnownas CountryName</FactReading>
      </FactReadings>
      <FactConstraints>
        <FactConstraint FactConstraintID="106" />
        <FactConstraint FactConstraintID="33" />
        <FactConstraint FactConstraintID="32" />
      </FactConstraints>
    </Fact>
    <Fact FactID="24" Arity="2" IsExternal="false" Storage="Stored">
      <FactRoles>
        <FactRole FactRoleID="22" />
        <FactRole FactRoleID="23" />
      </FactRoles>
      <FactReadings>
        <FactReading>Student bestAt Subject</FactReading>
      </FactReadings>
      <FactConstraints>
    <FactConstraint FactConstraintID="92" />
    <FactConstraint FactConstraintID="26" />
    <FactConstraint FactConstraintID="96" />
  </FactConstraints>
</Fact>
<Fact FactID="17" Arity="2" IsExternal="false" Storage="Stored">
  <FactRoles>
    <FactRole FactRoleID="15" />
    <FactRole FactRoleID="16" />
  </FactRoles>
  <FactReadings>
    <FactReading>Student represents Country</FactReading>
  </FactReadings>
  <FactConstraints>
    <FactConstraint FactConstraintID="89" />
    <FactConstraint FactConstraintID="20" />
    <FactConstraint FactConstraintID="96" />
  </FactConstraints>
</Fact>
<Fact FactID="44" Arity="2" IsExternal="false" Storage="Stored">
  <FactRoles>
    <FactRole FactRoleID="42" />
    <FactRole FactRoleID="43" />
  </FactRoles>
  <FactReadings>
    <FactReading>Student has Name</FactReading>
  </FactReadings>
  <FactConstraints>
    <FactConstraint FactConstraintID="86" />
    <FactConstraint FactConstraintID="46" />
  </FactConstraints>
</Fact>
<Fact FactID="37" Arity="2" IsExternal="false" Storage="Stored">
  <FactRoles>
    <FactRole FactRoleID="35" />
    <FactRole FactRoleID="36" />
  </FactRoles>
  <FactReadings>
    <FactReading>Subject means SubjectName</FactReading>
  </FactReadings>
  <FactConstraints>
    <FactConstraint FactConstraintID="103" />
    <FactConstraint FactConstraintID="40" />
    <FactConstraint FactConstraintID="39" />
  </FactConstraints>
</Fact>
<Fact FactID="57" Arity="2" IsExternal="false" Storage="Stored">
  <FactRoles>
    <FactRole FactRoleID="55" />
    <FactRole FactRoleID="56" />
  </FactRoles>
  <FactReadings>
    <FactReading>Country is identified by Countryid</FactReading>
  </FactReadings>
  <FactConstraints>
    <FactConstraint FactConstraintID="61" />
    <FactConstraint FactConstraintID="59" />
    <FactConstraint FactConstraintID="60" />
  </FactConstraints>
</Fact>
<Fact FactID="50" Arity="2" IsExternal="false" Storage="Stored">
  <FactRoles>
```

```
        <FactRole FactRoleID="48" />
        <FactRole FactRoleID="49" />
      </FactRoles>
      <FactReadings>
        <FactReading>Group contains Student</FactReading>
        <FactReading>Student attends Group</FactReading>
      </FactReadings>
      <FactConstraints>
        <FactConstraint FactConstraintID="52" />
        <FactConstraint FactConstraintID="100" />
      </FactConstraints>
    </Fact>
    <Fact FactID="73" Arity="2" IsExternal="false" Storage="Stored">
      <FactRoles>
        <FactRole FactRoleID="71" />
        <FactRole FactRoleID="72" />
      </FactRoles>
      <FactReadings>
        <FactReading>Subject is identified by Subjectid</FactReading>
      </FactReadings>
      <FactConstraints>
        <FactConstraint FactConstraintID="77" />
        <FactConstraint FactConstraintID="75" />
        <FactConstraint FactConstraintID="76" />
      </FactConstraints>
    </Fact>
    <Fact FactID="65" Arity="2" IsExternal="false" Storage="Stored">
      <FactRoles>
        <FactRole FactRoleID="63" />
        <FactRole FactRoleID="64" />
      </FactRoles>
      <FactReadings>
        <FactReading>Group is identified by Groupid</FactReading>
      </FactReadings>
      <FactConstraints>
        <FactConstraint FactConstraintID="69" />
        <FactConstraint FactConstraintID="67" />
        <FactConstraint FactConstraintID="68" />
      </FactConstraints>
    </Fact>
    <Fact FactID="80" Arity="2" IsExternal="false" Storage="Stored">
      <FactRoles>
        <FactRole FactRoleID="78" />
        <FactRole FactRoleID="79" />
      </FactRoles>
      <FactReadings>
        <FactReading>Group hasTopic Topic</FactReading>
      </FactReadings>
      <FactConstraints>
        <FactConstraint FactConstraintID="82" />
        <FactConstraint FactConstraintID="100" />
      </FactConstraints>
    </Fact>
  </Facts>
  <Constraints>
    <Constraint ConstraintID="26" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
    IsPrimaryReference="false">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="22" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="20" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
    IsPrimaryReference="false">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="15" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="46" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
    IsPrimaryReference="false">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="42" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="40" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
    IsPrimaryReference="false">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="35" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="39" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
    IsPrimaryReference="false">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="36" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="33" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
    IsPrimaryReference="false">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="28" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="32" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
    IsPrimaryReference="false">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="29" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="61" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Mandatory">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="55" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="60" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
    IsPrimaryReference="true">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="56" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="59" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
    IsPrimaryReference="false">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="55" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="52" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="2" ConstraintType="Uniqueness"
    IsPrimaryReference="false">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="48" />
            <RoleSequenceItem SequencePositionNumber="2"
            RoleSequenceItemRoleID="49" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="77" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Mandatory">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="71" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="76" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
    IsPrimaryReference="true">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="72" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="75" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
    IsPrimaryReference="false">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="71" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="69" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Mandatory">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="63" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="68" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
    IsPrimaryReference="true">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="64" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="67" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
```

```
IsPrimaryReference="false">
  <RoleSequences>
    <RoleSequence SequenceNumber="1">
      <RoleSequenceItems>
        <RoleSequenceItem SequencePositionNumber="1"
        RoleSequenceItemRoleID="63" />
      </RoleSequenceItems>
    </RoleSequence>
  </RoleSequences>
</Constraint>
<Constraint ConstraintID="92" IsInternal="true" NumberOfSequences="1"
NumberOfItemsPerSequence="1" ConstraintType="Mandatory">
  <RoleSequences>
    <RoleSequence SequenceNumber="1">
      <RoleSequenceItems>
        <RoleSequenceItem SequencePositionNumber="1"
        RoleSequenceItemRoleID="22" />
      </RoleSequenceItems>
    </RoleSequence>
  </RoleSequences>
</Constraint>
<Constraint ConstraintID="89" IsInternal="true" NumberOfSequences="1"
NumberOfItemsPerSequence="1" ConstraintType="Mandatory">
  <RoleSequences>
    <RoleSequence SequenceNumber="1">
      <RoleSequenceItems>
        <RoleSequenceItem SequencePositionNumber="1"
        RoleSequenceItemRoleID="15" />
      </RoleSequenceItems>
    </RoleSequence>
  </RoleSequences>
</Constraint>
<Constraint ConstraintID="86" IsInternal="true" NumberOfSequences="1"
NumberOfItemsPerSequence="1" ConstraintType="Mandatory">
  <RoleSequences>
    <RoleSequence SequenceNumber="1">
      <RoleSequenceItems>
        <RoleSequenceItem SequencePositionNumber="1"
        RoleSequenceItemRoleID="42" />
      </RoleSequenceItems>
    </RoleSequence>
  </RoleSequences>
</Constraint>
<Constraint ConstraintID="82" IsInternal="true" NumberOfSequences="1"
NumberOfItemsPerSequence="1" ConstraintType="Uniqueness"
IsPrimaryReference="false">
  <RoleSequences>
    <RoleSequence SequenceNumber="1">
      <RoleSequenceItems>
        <RoleSequenceItem SequencePositionNumber="1"
        RoleSequenceItemRoleID="78" />
      </RoleSequenceItems>
    </RoleSequence>
  </RoleSequences>
</Constraint>
<Constraint ConstraintID="106" IsInternal="true" NumberOfSequences="1"
NumberOfItemsPerSequence="1" ConstraintType="Mandatory">
  <RoleSequences>
    <RoleSequence SequenceNumber="1">
      <RoleSequenceItems>
        <RoleSequenceItem SequencePositionNumber="1"
        RoleSequenceItemRoleID="28" />
```

```
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="103" IsInternal="true" NumberOfSequences="1"
    NumberOfItemsPerSequence="1" ConstraintType="Mandatory">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="35" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="100" IsInternal="false" NumberOfSequences="1"
    NumberOfItemsPerSequence="2" ConstraintType="Uniqueness"
    IsPrimaryReference="false">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="49" />
            <RoleSequenceItem SequencePositionNumber="2"
            RoleSequenceItemRoleID="79" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
    <Constraint ConstraintID="96" IsInternal="false" NumberOfSequences="1"
    NumberOfItemsPerSequence="2" ConstraintType="Uniqueness"
    IsPrimaryReference="true">
      <RoleSequences>
        <RoleSequence SequenceNumber="1">
          <RoleSequenceItems>
            <RoleSequenceItem SequencePositionNumber="1"
            RoleSequenceItemRoleID="16" />
            <RoleSequenceItem SequencePositionNumber="2"
            RoleSequenceItemRoleID="23" />
          </RoleSequenceItems>
        </RoleSequence>
      </RoleSequences>
    </Constraint>
  </Constraints>
 </ORMSourceModel>
 </ORMSourceModels>
</VisioModels>
```

# APPENDIX C

# RDF Schema of OWL-K

```
<?xml version="1.0"?> <!DOCTYPE rdf:RDF [
     <!ENTITY rdf  "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
     <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
     <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
     <!ENTITY owlk "http://dieuthu.free.fr/2007/orm2owl/owlk#" >
   ]>

<rdf:RDF
  xmlns:rdf ="&rdf;"
  xmlns:rdfs="&rdfs;"
  xmlns:owl ="&owl;"
  xmlns     ="&owlk;"
  xml:base  ="&owlk;"
  xmlns:owlk ="&owlk;"
>

<owl:Ontology rdf:about="">
  <imports rdf:resource="http://www.w3.org/2000/01/rdf-schema"/>
  <rdfs:isDefinedBy rdf:resource="http://dieuthu.free.fr/2007/orm-owl/" />
  <rdfs:comment>This file specifies in RDF Schema format the
    built-in classes and properties that together form the basis of
    the RDF/XML syntax of the extension of OWL K w.r.t OWL DL.
    We do not expect people to import this file explicitly into their
    ontology. People that do import this file should expect their ontology
    to be an OWL Full ontology.
  </rdfs:comment>
  <versionInfo> June 2007 </versionInfo>
</owl:Ontology>
```

```
<rdfs:Class rdf:ID="ICAssertion">
  <rdfs:label>ICAssertion</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Resource"/>
</rdfs:Class>

<rdf:Property rdf:ID="onClass">
  <rdfs:label>onClass</rdfs:label>
  <rdfs:domain rdf:resource="#ICAssertion"/>
  <rdfs:range rdf:resource="#ObjectProperty"/>
</rdf:Property>

<rdf:Property rdf:ID="byProperty">
  <rdfs:label>byProperty</rdfs:label>
  <rdfs:domain rdf:resource="#ICAssertion"/>
  <rdfs:range rdf:resource="#ObjectProperty"/>
</rdf:Property>

</rdf:RDF>
```

APPENDIX D

# ORM Schema in OWL-K Ontology

This appendix shows the content of the OWL-K ontology which captures all the information expressed in the ORM schema example in chapter 7. Appendix D.1 shows the ontology in abstract syntax. Appendix D.2 shows the ontology in RDF/XML syntax.

## D.1 OWL-K Ontology in Abstract syntax

```
Namespace(rdf   = http://www.w3.org/1999/02/22-rdf-syntax-ns#)
Namespace(xsd = http://www.w3.org/2001/XMLSchema#)
Namespace(rdfs  = http://www.w3.org/2000/01/rdf-schema#)
Namespace(owl   = http://www.w3.org/2002/07/owl#)
Namespace(owlk  = http://dieuthu.free.fr/2007/orm2owl/owlk#)

Ontology(
    Annotation(rdfs:comment "An OWL ontology from an ORM schema")

Class(Subject partial)
Class(Country partial)
Class(Student partial)
Class(Group partial)

DatatypeProperty(is_identified_by_63 domain(Group) range(Groupid))

ICAssertion(Group_51 Group is_identified_by63)
DatatypeProperty(is_identified_by_55 domain(Country) range(Countryid))

ICAssertion(Country_19 Country is_identified_by55)

DatatypeProperty(is_identified_by_71 domain(Subject) range(Subjectid))

ICAssertion(Subject_25 Subject is_identified_by71)

DatatypeProperty(isKnownas Domain(Country) Range(CountryName))

ObjectProperty(bestAt Domain(Student) Range(Subject))

ObjectProperty(represents Domain(Student) Range(Country))

DatatypeProperty(has Domain(Student) Range(Name))

DatatypeProperty(means Domain(Subject) Range(SubjectName))

ObjectProperty(contains Domain(Group) Range(Student))

ObjetcProperty(attends inverseOf(contains))

DatatypeProperty(hasTopic Domain(Group) Range(Topic))

ICAssertion(Country_28 Country isKnownas)

ICAssertion(Subject_35 Subject means)

ICAssertion(Student_18 Student represents bestAt)

Class(Student partial restriction(has someValuesFrom(Name)))

Class(Student partial restriction(has maxCardinality(1)))

Class(Group partial restriction(hasTopic maxCardinality(1)))

ObjectProperty(contains_hasTopic_108
      domain(Student_Topic_107) range(Group))

Class(Student_Topic_107 partial
      restriction(contains_hasTopic_108 maxCardinality(1)))

ICAssertion(Student_Topic_107_107 Student_Topic_107 contains hasTopic)
)
```

## D.2 OWL-K Ontology in RDF/XML syntax

This appendix presents the RDF/XML serialization of the OWL-K ontology that describes the ORM schema in Figure 7.1.

```
<rdf:RDF
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:owlk="http://dieuthu.free.fr/2007/orm2owl/owlk#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
>

 <owl:Ontology rdf:about="">
    <rdfs:comment>An OWL ontology from an ORM schema</rdfs:comment>
 </owl:Ontology>

 <owl:Class rdf:ID="Subject"></owl:Class>
 <owl:Class rdf:ID="Country"></owl:Class>
 <owl:Class rdf:ID="Student"></owl:Class>
 <owl:Class rdf:ID="Group"></owl:Class>

 <owl:DatatypeProperty rdf:ID="is_identified_by_63">
    <rdfs:domain rdf:resource="#Group"></rdfs:domain>
    <rdfs:range rdf:resource="#Groupid"></rdfs:range>
 </owl:DatatypeProperty>

 <owlk:ICAssertion rdf:ID="Group_51">
    <owlk:onClass rdf:resource="#Group"/>
    <owlk:byProperty rdf:resource="#is_identified_by63"/>
 </owlk:ICAssertion>

 <owl:DatatypeProperty rdf:ID="is_identified_by_55">
    <rdfs:domain rdf:resource="#Country"></rdfs:domain>
    <rdfs:range rdf:resource="#Countryid"></rdfs:range>
 </owl:DatatypeProperty>

 <owlk:ICAssertion rdf:ID="Country_19">
    <owlk:onClass rdf:resource="#Country"/>
    <owlk:byProperty rdf:resource="#is_identified_by55"/>
 </owlk:ICAssertion>

 <owl:DatatypeProperty rdf:ID="is_identified_by_71">
    <rdfs:domain rdf:resource="#Subject"></rdfs:domain>
    <rdfs:range rdf:resource="#Subjectid"></rdfs:range>
 </owl:DatatypeProperty>
```

```
<owlk:ICAssertion rdf:ID="Subject_25">
   <owlk:onClass rdf:resource="#Subject"/>
   <owlk:byProperty rdf:resource="#is_identified_by71"/>
</owlk:ICAssertion>

<owl:DatatypeProperty rdf:ID="isKnownas">
   <rdfs:domain rdf:resource="#Country"></rdfs:domain>
   <rdfs:range rdf:resource="#CountryName"></rdfs:range>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="bestAt">
   <rdfs:domain rdf:resource="#Student"></rdfs:domain>
   <rdfs:range rdf:resource="#Subject"></rdfs:range>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="represents">
   <rdfs:domain rdf:resource="#Student"></rdfs:domain>
   <rdfs:range rdf:resource="#Country"></rdfs:range>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="has">
   <rdfs:domain rdf:resource="#Student"></rdfs:domain>
   <rdfs:range rdf:resource="#Name"></rdfs:range>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="means">
   <rdfs:domain rdf:resource="#Subject"></rdfs:domain>
   <rdfs:range rdf:resource="#SubjectName"></rdfs:range>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="contains">
   <rdfs:domain rdf:resource="#Group"></rdfs:domain>
   <rdfs:range rdf:resource="#Student"></rdfs:range>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="attends">
   <owl:inverseOf rdf:resource="#contains">
</owl:inverseOf>

<owl:DatatypeProperty rdf:ID="hasTopic">
   <rdfs:domain rdf:resource="#Group"></rdfs:domain>
   <rdfs:range rdf:resource="#Topic"></rdfs:range>
</owl:DatatypeProperty>

<owlk:ICAssertion rdf:ID="Country_28">
   <owlk:onClass rdf:resource="#Country"/>
   <owlk:byProperty rdf:resource="#isKnownas"/>
</owlk:ICAssertion>
```

```
<owlk:ICAssertion rdf:ID="Subject_35">
   <owlk:onClass rdf:resource="#Subject"/>
   <owlk:byProperty rdf:resource="#means"/>
</owlk:ICAssertion>

<owlk:ICAssertion rdf:ID="Student_18">
   <owlk:onClass rdf:resource="#Student"/>
   <owlk:byProperty rdf:resource="#represents"/>
   <owlk:byProperty rdf:resource="#bestAt"/>
</owlk:ICAssertion>

<owl:Class rdf:about="#Student">
       <rdfs:subClassOf>
           <owl:Restriction>
               <owl:onProperty rdf:resource="#has"/>
               <owl:someValuesFrom rdf:resource="#Name"/>
           </owl:Restriction>
       </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Student">
       <rdfs:subClassOf>
           <owl:Restriction>
               <owl:onProperty rdf:resource="#has"/>
               <owl:maxCardinality rdf:datatype=
               "http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
               1</owl:maxCardinality>
           </owl:Restriction>
       </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Group">
       <rdfs:subClassOf>
           <owl:Restriction>
               <owl:onProperty rdf:resource="#hasTopic"/>
               <owl:maxCardinality rdf:datatype=
               "http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
               1</owl:maxCardinality>
           </owl:Restriction>
       </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="contains_hasTopic_108">
   <rdfs:domain rdf:resource="#Student_Topic_107"></rdfs:domain>
   <rdfs:range rdf:resource="#Group"></rdfs:range>
</owl:ObjectProperty>

<owl:Class rdf:about="#Student_Topic_107">
       <rdfs:subClassOf>
```

```
            <owl:Restriction>
                <owl:onProperty rdf:resource="#contains_hasTopic_108"/>
                <owl:maxCardinality rdf:datatype=
                "http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
                1</owl:maxCardinality>
            </owl:Restriction>
        </rdfs:subClassOf>
 </owl:Class>

 <owlk:ICAssertion rdf:ID="Student_Topic_107_107">
    <owlk:onClass rdf:resource="#Student_Topic_107"/>
    <owlk:byProperty rdf:resource="#contains"/>
    <owlk:byProperty rdf:resource="#hasTopic"/>
 </owlk:ICAssertion>

</rdf:RDF>
```

# Bibliography

[1] Jena framework. http://jena.sourceforge.net/.

[2] Neumont ORM Architect for Visual Studio. Available at http://sourceforge.net/projects/orm.

[3] OMG, UML Notation Guide, Version 1.1, 1997. Available at http://dblp.l3s.de/d2r/resource/publications/www/org/omg/uml-notation-guide.

[4] H. Alvestrand. RFC 3006 Tags for the Identification of Languages. Available at http://www.isi.edu/in-notes/rfc3066.txt.

[5] F. Baader. Augmenting Concept Languages by Transitive Closure of Roles: An Alternative to Terminological Cycles. Technical Report RR-90-13, Deutsches Forschungszentrum für Künstliche Intelligenz GmbHErwin-Schrödinger Strasse, Germany, 1990.

[6] F. Baader. Augmenting Concept Languages by Transitive Closure of Roles: An Alternative to Terminological Cycles. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI-91*, pages 446–451, Sydney (Australia), 1991.

[7] F. Baader, D. Calvanese, Deborah L. McGuinness, Daniele N., and Peter F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications.* Cambridge University Press, New York, NY, USA, 2003.

[8] F. Baader, B. Hollunder, B. N., Hans-Jürgen Profitlich, and Enrico Franconi. An Empirical Analysis of Optimization Techniques for Terminological Representation Systems or “Making KRIS get a move on. In B. Nebel, W. Swartout, and C. Rich, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 3rd International Conference*, pages 270–281, San Mateo, 1992. Morgan Kaufmann.

[9] F. Baader and Werner Nutt. Basic description logics. In *The description logic handbook: theory, implementation, and applications*, pages 43–95. Cambridge University Press, New York, NY, USA, 2003.

[10] G.P. Bakema, J.P.C. Zwart, and H. van der Lek. Fully Communication Oriented NIAM. In *Proceedings of NIAM-ISDM 2*, pages 1–35, Albuquerque, New Mexico, 1994.

[11] Francois Barbancon and Daniel P. Miranker. Implementing Federated Databases Systems by Compiling SchemaSQL. In *IDEAS ’02: Proceedings of the 2002 International Symposium on Database Engineering & Applications*, pages 192–201, Washington, DC, USA, 2002. IEEE Computer Society.

[12] Scot A. Becker. Normalization and ORM. *The Journal of Conceptual Modeling*, Issue 4, 1998.

[13] Scot A. Becker. Data Schema Normalization. *Journal of Conceptual Modeling*, Issue 9, 1999.

[14] Scot A. Becker. Perspective and Abstraction. *Journal of Conceptual Modeling*, Issue 17, December 2000.

[15] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1):70–118, 2005.

[16] Berners-Lee. Relational Databases on the Semantic Web, 1998. Available at http://www.w3.org/DesignIssues/RDB-RDF.html.

[17] Tim Berners-Lee. Semantic Web - XML2000. 2000. Available at http://www.w3.org/2000/Talks/1206-xml2k-tbl/.

[18] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web: Scientific American. *Scientific American*, May 2001.

[19] Christian Bizer. D2R MAP - A Database to RDF Mapping Language. In *12th International World Wide Web Conference, Budapest, May*, 2003.

[20] Christian Bizer and Andy Seaborne. D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs. In *ISWC2004 (posters)*, November 2004.

[21] Anthony C. Bloesch and Terry A. Halpin. ConQuer: A Conceptual Query Language. In *ER '96: Proceedings of the 15th International Conference on Conceptual Modeling*, pages 121–133, London, UK, 1996. Springer-Verlag.

[22] Peter Bollen. A Formal ORM-to-UML Mapping Algorithm. Research Memoranda 015, Maastricht : METEOR, Maastricht Research School of Economics of Technology and Organization, 2002. Available at http://ideas.repec.org/p/dgr/umamet/2002015.html.

[23] A. Borgida and Grant E. Weddell. Adding Uniqueness Constraints to Description Logics (Preliminary Report). In *DOOD '97: Proceedings of the 5th International Conference on Deductive and Object-Oriented Databases*, pages 85–102, London, UK, 1997. Springer-Verlag.

[24] Ronald J. Brachman and Hector J. Levesque. The Tractability of Subsumption in Frame-Based Description Languages. In *Proceedings of the 4th National Conference on Artificial Intelligence (AAAI'84)*, pages 34–37, 1984.

[25] Dan Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0, 2000. http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

[26] Agustina Buccella, Miguel R. Penabad, Francisco J. Rodriguez, Antonio Faria, and Alejandra Cechich. From relational Databases to OWL ontologies. In *Sixth National Russian Research Conference*, 2004. September 29 - October 1.

[27] D. Calvanese, G. DeGiacomo, and M. Lenzerini. Conjunctive query containment in description logics with n-ary relations. In *International Workshop on Description Logics, Paris, France*, 1997.

[28] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Structured Objects: Modeling and Reasoning. In *Proceedings of the 4th International Conference on Deductive and Object-Oriented Databases (DOOD'95)*, number 1013 in Lecture Notes in Computer Science, pages 229–246. Springer, 1995.

[29] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Identification Constraints and Functional Dependencies in Description Logics. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 155–160, 2001.

[30] Diego Calvanese and Giuseppe De Giacomo. Data integration: a logic-based perspective. *AI Magazine*, 26(1):59–70, 2005.

[31] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the Decidability of Query Containment under Constraints. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, pages 149–158. ACM Press, 1998.

[32] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. A Framework for Ontology Integration. In *The Emerging Semantic Web, Selected papers from the first Semantic web working symposium, Stanford University, California, USA, July 30 - August 1*, 2001.

[33] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Information Integration: Conceptual Modeling and Reasoning Support. In *Conference on Cooperative Information Systems*, pages 280–291, 1998.

[34] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: Implementing the semantic web recommendations. Technical Report HPL-2003.

[35] T. Catarci and M. Lenzerini. Representing and Using Interschema Knowledge in Cooperative Information Systems. *Journal for Intelligent and Cooperative Information Systems*, 2(4):375–399, 1993.

[36] Huajun Chen, Yimin Wang, Heng Wang, Yuxin Mao, Jinmin Tang, Cunyin Zhou, Ainin Yin, and Zhaohui Wu. Towards a Semantic Web of Relational Databases: a Practical Semantic Toolkit and an In-Use Case from Traditional Chinese Medicine. In *4th International Semantic Web Conference (ISWC'06)*, LNCS, pages 750–763, Athens, USA, November 2006. Springer-Verlag. Best Paper Award.

[37] Peter Pin-Shan S. Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[38] Clark and Parsia LLC. Pellet: The Open Source OWL DL Reasoner. http://pellet.owldl.com/.

[39] Intellidimension Company. RDF Gateway, 2000. Available at http://www.intellidimension.com.

[40] Microsoft Corporation. Microsoft Visio for Enterprise Architects (VEA). http://www.microsoft.com/technet/prodtechnol/visio/visio2002/plan/ visdbmdl.mspx, http://msdn2.microsoft.com/es-es/library/ms182014.aspx.

[41] Giuseppe De Giacomo and Maurizio Lenzerini. What's in an aggregate: Foundations for description logics with tuples and sets. In Chris Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 801–807, San Francisco, 1995. Morgan Kaufmann.

[42] Cristian Pérez de Laborda and Stefan Conrad. Relational.OWL: a data and schema representation format based on OWL. In *APCCM '05: Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling*, pages 89–96, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[43] Jan Demey, Mustafa Jarrar, and Robert Meersman. A Markup Language for ORM Business Rules. In *Rule Markup Languages for Business Rules on the Semantic Web, in conjunction with the First International Semantic Web Conference*, 2002.

[44] AnHai Doan and Alon Y. Halevy. Semantic Integration Research in the Database Community: A Brief Survey. *AI Magazine*, 26:83–94, 2005.

[45] Francesco M. Donini. Complexity of reasoning. In *The description logic handbook: theory, implementation, and applications*, pages 96–136. Cambridge University Press, New York, NY, USA, 2003.

[46] Dejing Dou, Paea LePendu, Shiwoong Kim, and Peishen Qi. Integrating Databases into the Semantic Web through an Ontology-Based Framework. In *ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06)*, page 54, Washington, DC, USA, April 2006. IEEE Computer Society.

[47] Ramez A. Elmasri and Shankrant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[48] David W. Embley, Barry D. Kurtz, and Scott N. Woodfield. *Object-oriented systems analysis: a model-driven approach*. Yourdon Press, Upper Saddle River, NJ, USA, 1992.

[49] Andy S. Evans. Reasoning with UML Class Diagrams. In *WIFT '98: Proceedings of the Second IEEE Workshop on Industrial Strength Formal Specification Techniques*, page 102, Washington, DC, USA, 1998. IEEE Computer Society.

[50] Eckhard D. Falkenberg. Concepts for Modelling Information. In *IFIP Working Conference on Modelling in Data Base Management Systems*, pages 95–109. ed. G.M. Nijssen, Freudenstadt, Germany, North-Holland Publishing, 1976.

[51] Dieter Fensel, Ian Horrocks, Frank van Harmelen, Deborah L. McGuinness, and Peter F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2), 2001.

[52] Melvin Fitting. *First-order logic and automated theorem proving.* Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[53] Robert France, Andy Evans, and Kevin Lano. The UML as a Formal Modeling Notation. In Haim Kilov, Bernhard Rumpe, and Ian Simmonds, editors, *Proceedings OOPSLA'97 Workshop on Object-oriented Behavioral Semantics*, pages 75–81. Technische Universität München, TUM-I9737, 1997.

[54] Herve Gallaire and Jack Minker. *Logic and Databases.* Plenum Press, New York, USA, 1978.

[55] Hector Garcia-Molina, Yannis Papakonstantinou, Dallan Quass, Anand Rajaraman, Yehoshua Sagiv, Jeffrey D. Ullman, Vasilis Vassalos, and Jennifer Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[56] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer? *WebDB Workshop on Databases and the Web*, June 2001.

[57] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[58] G. Gupta. Horn Logic Denotations and Their Applications. In *Proceedings of Workshop on Current trends and Future Directions in Logic Programming Research, April*, 1998.

[59] V. Haarslev and R. Möller. Description of the RACER System and its Applications. In *Proceedings International Workshop on Description Logics (DL-2001), Stanford, USA, 1.-3. August*, pages 131–141, 2001.

[60] V. Haarslev and R. Möller. Racer: A Core Inference Engine for the Semantic Web. In *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), located at the 2nd International Semantic Web Conference ISWC 2003, Sanibel Island, Florida, USA, October 20*, pages 27–36, 2003.

[61] H. Habrias. Normalized Object Oriented Method. In *Encyclopedia of Microcomputers*, volume 12, pages 271–285. Marcel Dekker, New York, USA, 1993.

[62] Ives-Z.G. Madhavan J. Mork P. Suciu D. Halevy, A.Y. and I. Tatarinov. The Piazza Peer Data Management System. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):787–798, 2004.

[63] Y. Halevy, G. Ives, Dan Suciu, and Igor Tatarinov. Schema mediation for large-scale semantic data sharing. *The VLDB Journal*, 14(1):68–83, 2005.

[64] TA Halpin. Object Role Modeling, The official site for Conceptual Data Modeling. Available at http://www.orm.net/.

[65] TA Halpin. *A logical analysis of information systems: static aspects of the data-oriented perspective.* PhD thesis, University of Queensland, Australia, 1989.

[66] TA. Halpin. Augmenting UML with Fact Orientation. In *HICSS*, 2001.

[67] TA. Halpin. *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[68] TA Halpin. Object-role modeling (ORM/NIAM). In *Handbook on Architectures of Information Systems, 2nd edition*, pages 81–103. Springer, Heidelberg, 2006.

[69] TA Halpin. ORM 2 graphical notation. Technical report, Newmont University, September 2005.

[70] D. Harel and B. Rumpe. Modeling Languages: Syntax, Semantics and All That Stuff, Part I: The Basic Stuff. Technical report, Jerusalem, Israel, 2000. Weizmann Science Press of Israel.

[71] Patrick Hayes. RDF Semantics. Technical report. Available at http://www.w3.org/TR/rdf-mt/.

[72] Jochen Heinsohn, Daniel Kudenko, Bernhard Nebel, and Hans-Jürgen Profitlich. An Empirical Analysis of Terminological Representation Systems. *Artificient Intelligence*, 68(2):367–397, 1994.

[73] B. Hollunder and F. Baader. Qualifying Number Restrictions in Concept Languages. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning, KR-91*, pages 335–346, Boston (USA), 1991.

[74] Rector A. Stevens R. Wroe C. Horridge M., Knublauch H. A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and COODE Tools Edition 1.0, 2004. The University Of Manchester Stanford University.

[75] I. Horrocks. The FaCT System. Available at http://www.cs.man.ac.uk/ horrocks/FaCT/.

[76] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[77] I. Horrocks. DAML+OIL: A Reasonable Web Ontology Language. In *Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology, Prague, Czech Republic, March 25-27, Proceedings*, volume 2287 of *Lecture Notes in Computer Science*, pages 2–13. Springer, 2002.

[78] I. Horrocks. Implementation and optimization techniques. In *The description logic handbook: theory, implementation, and applications*, pages 306–346. Cambridge University Press, New York, NY, USA, 2003.

[79] I. Horrocks, P. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[80] I. Horrocks and Peter F. Patel-Schneider. The Generation of DAML+OIL. In *the 2001 International Description Logics Workshop (DL-2001), Stanford, CA, USA*, volume 49 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2001.

[81] I. Horrocks and Peter F. Patel-Schneider. OWL Web Ontology Language, Semantics and Abstract Syntax, 2004. Available at http://www.w3.org/TR/2004/REC-owl-semantics-20040210/syntax.html.

[82] I. Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Journal of Web Semantics*, 1(4):345–357, 2004.

[83] I. Horrocks and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, Los Altos, 2001.

[84] I. Horrocks and U. Sattler. A Tableaux Decision Procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 448–453, 2005.

[85] I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies. How to Decide Query Containment Under Constraints Using a Description Logic. In *7th International Workshop on Knowledge Representation meets Databases (KRDB2000)*, 2000.

[86] I. Horrocks, U. Sattler, and S. Tobies. A Description Logic with Transitive and Converse Roles, Role Hierarchies and Qualifying Number Restrictions. LTCS-Report 99-08, LuFg Theoretical Computer Science, RWTH Aachen, Germany, 1999.

[87] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In *LPAR '99: Proceedings of the 6th International Conference on Logic Programming and Automated Reasoning*, pages 161–180, London, UK, 1999. Springer-Verlag.

[88] Richard Hull and Roger King. Semantic database modeling: survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[89] Richard Hull and Gang Zhou. A framework for supporting data integration using the materialized and virtual approaches. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 481–492, New York, NY, USA, 1996. ACM Press.

[90] Mustafa Jarrar. Towards Automated Reasoning on ORM Schemes. -Mapping ORM into the $\mathcal{DLR}_{idf}$ description logic. In *Proceedings of the 26th International Conference on Conceptual Modeling (ER 2007)*, Auckland, New Zealand, November 2007. Springer.

[91] W. Kent. Entities and relationships in Information. In *IFIP Working Conference on Modelling in Data Base Management Systems*, Nice, France, 1977.

[92] Vitaliy L. Khizder, David Toman, and Grant E. Weddell. On decidability and complexity of description logics with uniqueness constraints. In *Description Logics*, pages 193–202, 2000.

[93] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42(4):741–843, 1995.

[94] Donald E. Knuth. Backus normal form vs. Backus Naur form. *Commun. ACM*, 7(12):735–736, 1964.

[95] Maurizio Lenzerini. Data integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM.

[96] C. Lutz. Complexity of Terminological Reasoning Revisited. In *LPAR '99: Proceedings of the 6th International Conference on Logic Programming and Automated Reasoning*, pages 181–200, London, UK, 1999. Springer-Verlag.

[97] C. Lutz, C. Areces, I. Horrocks, and U. Sattler. Keys, Nominals, and Concrete Domains. *Journal of Artificial Intelligence Research*, 23:667–7263, 2005.

[98] Alexander Maedche, Boris Motik, Nuno Silva, and Raphael Volz. MAFRA - A MApping FRAmework for Distributed Ontologies. In *EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, pages 235–250, London, UK, 2002. Springer-Verlag.

[99] F. Manola and E. Miller. RDF Primer, 2004. http://www.w3.org/TR/REC-rdf-syntax/.

[100] Drew V. McDermott. The Planning Domain Definition Language Manual. Technical Report CVC Report 98-003, Yale Computer Science, 1998.

[101] Drew V. McDermott and Dejing Dou. Representing Disjunction and Quantifiers in RDF. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 250–263, London, UK, 2002. Springer-Verlag.

[102] R. Meersman. The RIDL conceptual language. Research report, Int. Centre for Information Analysis Services, Control Data, Brussels, Belgium, 1982.

[103] Microsoft download Center. Microsoft Corporation. Visio 2000 Tool: VisioModeler (Unsupported Product Edition). http://www.microsoft.com/downloads/details.aspx.

[104] Robert J. Muller. *Database Design for Smarties: Using UML for Data Modeling.* Morgan Kaufmann, San Francisco, CA, USA, February 1999.

[105] Bernhard Nebel. *Reasoning and revision in hybrid representation systems.* Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[106] Thi Dieu Thu Nguyen and Nhan Le-Thanh. Identification constraints in $\mathcal{SHOIN}(\mathbf{D})$. In *Proceedings of the 1th International Conference on Research Challeges in Information Science (RCIS 2007)*, Ouarzazate, Morocco, April 2007.

[107] Thi Dieu Thu Nguyen and Nhan Le-Thanh. Integrating Identification Constraints in Web Ontology. In *Proceedings of the 9th International Conference on Enterprise Information Systems*, Madeira, Portugal, June 2007.

[108] Thi Dieu Thu Nguyen, Nhan Le-Thanh, and Anh Le Pham. Modeling ORM Schemas in Description Logics. In *Proceedings of the 14th ISPE International Conference on Concurrent Engineering (CE2007)*, Sao Jos dos Campos, Brazil, July 2007.

[109] G. M. Nijssen. A Gross Architecture for the Next Generation Database Management System. In *IFIP Working Conference on Modelling in Data Base Management Systems*, pages 1–24, 1976.

[110] G. M. Nijssen. Current issues in conceptual schema concepts. In *IFIP Working Conference on Modelling in Data Base Management Systems*, Nice, France, 1977.

[111] P. E. van der Vet P.-H. Speel, F. van Raalte and N. J. I. Mars. Runtime and memory usage performance of description logics. In *Knowledge Retrieval, Use and Storage for Efficiency: Proceedings of the First International KRUSE Symposium*, pages 13–27, 1995.

[112] Jeff Z. Pan. *Description Logics: Reasoning Support for the Semantic Web.* PhD thesis, School of Computer Science, The University of Manchester, Oxford Rd, Manchester M13 9PL, UK, 2004.

[113] James Rumbaugh, Ivar Jacobson, and Grady Booch, editors. *The Unified Modeling Language reference manual.* Addison-Wesley Longman Ltd., Essex, UK, UK, 1999.

[114] J. Hendler I.Horrocks D.L. McGuinness P.F. Patel-Schneider S. Bechhofer, F. van Harmelen and L.A. Stein. OWL Web Ontology Language Reference, 2003. http://www.w3.org/TR/2003/CR-owl-ref-20030818/.

[115] Klaus Schild. A correspondence theory for terminological logics: preliminary report. In *Proceedings of IJCAI-91, 12th International Joint Conference on Artificial Intelligence*, pages 466–471, Sidney, AU, 1991.

[116] M. Schmidt-Schau and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

[117] Amit P. Sheth and James A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Survey*, 22(3):183–236, 1990.

[118] Peretz Shoval and Nili Shreiber. Database Reverse Engineering: From the Relational to the Binary Relationship model. *Data Knowledge Engineering*, 10:293–315, 1993.

[119] Nuno Silva and ao Rocha Jo˙ Semantic Web Complex Ontology Mapping. In *WI '03: Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence*, page 82, Washington, DC, USA, 2003. IEEE Computer Society.

[120] Orthogonal software corporation. Orthogonal Toolbox. http://www.orthogonalsoftware.com/.

[121] Sergejus Sosunovas and Olegas Vasilecas. Tool-Supported Method for the Extraction of OCL from ORM Models. In Witold Abramowicz, editor, *Business Information Systems*, volume 4439 of *Lecture Notes in Computer Science*, pages 449–463. Springer, 2007.

[122] J. F. Sowa, editor. *Principles of Semantic Networks: Explorations in the Representation of Knowledge.* Morgan Kaufmann, San Mateo CA, 1991.

[123] P Spyns, SV Acker, M Wynants, M Jarrar, and A Lisovoy. Using a Novel ORM-Based Ontology Modelling Method to Build an Experimental Innovation Router. In *EKAW*, pages 82–98, 2004.

[124] Ljiljana Stojanovic, Nenad Stojanovic, and Raphael Volz. Migrating data-intensive web sites into the semantic web. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 1100–1107, New York, NY, USA, 2002. ACM.

[125] W3C Member Submission. OWL 1.1 Web Ontology Language Overview. http://www.w3.org/Submission/owl11-overview/.

[126] J. Yuan J. Sawaya M. Uschold T. Adams T. Barrett, D. Jones and D. Folger. RDF Representation of Metadata for Semantic Integration of Corporate Information Resource. In *Proceedings of Real World RDF and Semantic Web Applications Workshop held in conjunction with WWW-2002*, 2002.

[127] C. M. Sperberg-McQueen T. Bray, J. Paoli and E. Maler. Extensible markup language (xml) 1.0 (second edition), 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[128] J.H. ter Bekke. Comparative Study of Four Data Modeling Approaches. In *Proceedings 2nd international EMMSAD workshop, Barcelona*, 1993.

[129] Arthur H. M. ter Hofstede, Henderik Alex Proper, and Theo P. van der Weide. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):489–523, 1993.

[130] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation.* PhD thesis, RWTH Aachen, Germany, 2001.

[131] Olga De Troyer. A formalization of the binary object-role model based on logic. *Data Knowledge Engineering*, 19(1):1–37, 1996.

[132] Olga De Troyer and Robert Meersman. A Logic Framework for a Semantics of Object-Oriented Data Modeling. In *OOER '95: Proceedings of the 14th International Conference on Object-Oriented and Entity-Relationship Modelling*, pages 238–249, London, UK, 1995. Springer-Verlag.

[133] Dmitry Tsarkov, Ian Horrocks, and Peter F. Patel-Schneider. Optimizing Terminological Reasoning for Expressive Description Logics. *Journal of Automated Reasoning*, 39(3):277–316, 2007.

[134] Jeffrey D. Ullman, Hector Garcia-Molina, and Jennifer Widom. *Database Systems: The Complete Book.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[135] Patrick v. Bommel, Arthur H. M. ter Hofstede, and Theo P. van der Weide. Semantics and verification of object-role models. *Information Systems*, 16(5):471–495, 1991.

[136] Paul V.Biron and Malhotra. Extensible Markup Language (XML) Schema Part 2: Datatypes Second Edition. Available at http://www.w3.org/TR/xmlschema-2/.

[137] USA Visio Corporation. Guide to FORML, 1998.

[138] Raphael Volz, Daniel Oberle, Steffen Staab, and Rudi Studer. OntoLift prototype. Technical Report D11, WonderWeb project deliverable, 2002. Available at http://wonderweb.man.ac.uk/deliverables/documents/D11.pdf.

[139] JJVR Wintraecken. *The NIAM Information Analysis Method: theory and practice.* Kluwer Academic Publishers, 1990.