Faculty of Computer Science

European Master's Program In Computational Logic

Master Thesis

A Case Study on Intelligent Data Querying

Author:
X. Isabel K.
Juárez-Castro

Supervisor:
Prof. Enrico
Franconi

March 18, 2015

Contents

List of Figures

List of Tables

Abstract

We designed a new ontology based on wine data obtained from Wine.com. We first obtained the ORM2 conceptual model with a methodology which uses the constraints of the data to identify the relations and entities existing in it. To map this model to OWL 2 we introduced a new fragment of ORM2 for binary relations called ORM2^{bin} with formal DL semantics. We also created a semantic preserving mapping from this fragment to OWL 2. Finally we validated our ontology using the query verbalisations provided by Quelo.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text. In confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Bolzano, March 2015
X. Isabel K. Juárez-Castro

Chapter 1

Introduction

The world is complex. Most of the times one focuses only in a part of it, the domain of interest. Formalisms from different areas exist to model and represent the domain of interest. From the database area a strong formalism is Object-Role Modeling, ORM. With ORM one can represent the conceptual model graphically. ORM also provides a procedure to translate a conceptual schema to a relational schema [12].

But data stored in databases is not easily accessible. One has to know beforehand the structure of the database and the query language to reach the information. On the Semantic Web, on the other hand, data is stored in almost schemaless ontologies. With a few number of built-ins these ontologies are able to model diverse domains of interest. But one still has to know the query language, and so the need for ontology exploration is born.

Natural Language Interfaces (NLI) can pave the way for an easier access to data. Users query using Natural Language (NL), and the NLI translates it into the query language. One of such tools is Quelo [9]. Quelo is a tool for ontology exploration where the user creates NL queries using menus. However, this exploration is limited to the concepts and relations of the ontology. The data takes no active part in the formation of the query. For example, Quelo is unable to form queries such as "I am looking for wines made in *France*.".

1.1 Motivation and objectives

The purpose of this thesis is to design and create an ontology from real raw data to be used by Quelo. We proceeded in two stages. First, using as input the raw data we obtain its conceptual schema. Then, from the conceptual schema we obtain the ontology. This ontology is then validated using Quelo.

For the first stage, we use a methodology proposed by David et al. [4] and obtain an ORM model. The data used in this stage was obtained from Wine.com.

Wine.com is the biggest online wine retailer in the U.S. selling thousands of wine bottles per year. Wine.com has an inventory of over a million bottles, and in their web page they also provide with wine related information such as winemaker notes, ratings, customer reviews, region descriptions, and maps. Besides selling wines, they offer a public API to access their wine catalogue and

wine information [38]. Of this data we selected the data of the Italian region and some wines of France, Hungary and the U.S.

For the second stage, we translate the ORM model into an OWL 2 (Ontology Web Language) ontology. There are other works [20, 37, 26] which propose mappings from ORM to OWL. Hodrob and Jarrar [20] proposed a translation from ORM into the XML/OWL syntax, Wagih et al. [37] translated ORM2 into the Manchester syntax and Løvdahl [26] into the Turtle syntax. There are also some articles by T. Halpin [14, 15, 16, 17] which show examples of several ORM2 diagrams with their translation into Turtle and Manchester syntax. However, most of these works do not rely in a formal setting. Hodrob and Jarrar [20] use the $\mathcal{SHOIN}$ description logic; nevertheless, this proposal lacks formality and some of their translations are incorrect. Therefore, we propose a new mapping from a subset of ORM2 to OWL 2.

Finally, we validate the ontology by creating verbalisations of queries using Quelo. Since Quelo allows the formulation of only those queries satisfiable in an ontology, any query formulated which is not consistent with the domain of interest indicates a flaw on the design of the ontology.

1.2 Thesis outline

This thesis is structured as follows. In Chapter 2 we introduce the ORM2 graphical notation, describe the methodology of [4] to obtain the conceptual schema, and we follow this methodology using data from the wine domain. The second stage and the mapping from ORM2 to OWL 2 is described in Chapter 3. Chapter 4 gives a proper introduction to Quelo and presents the validation of the ontology. In the last chapter, Chapter 5, we summarize our results and discuss future work.

Chapter 2

From Raw Data to Conceptual Schema

In this chapter we describe the methodology followed to obtain the conceptual schema from the raw data. We also include an introduction to Object Role Modeling (ORM) in Section 2.1. In Section 2.2 we introduce the constraints used to restrict the data. We describe the data in Section 2.3. In Section 2.4 we outline the constraints found on the raw data. In Section 2.5 we explain the methodology proposed by David et al. [4] and how we applied it to our use case.

2.1 Object Role Modeling

Object Role Modeling (ORM) is a method to design conceptual models for databases. The current version of ORM is ORM2[13] and it is the one we will use in this thesis. ORM2 was provides with a graphical notation for the representation of the conceptual model and a method to obtain the relational schema from it. In this section we will introduce the graphical notation of ORM2.

2.1.1 ORM2 elements

The basic building blocks of an ORM2 diagram are, not surprisingly, *object types* and *roles*. Individuals of the same kind are grouped into object types [5]. There are two sorts of object types, *entity types* and *value types*. Sequences of roles form *predicates*, which associate object types. ORM2 also provides a notation to outline restrictions on both the object types and the roles.

Entity types represent concepts in ORM2, which are depicted with a named soft rectangle by default. The entity types can have a *reference mode*, the identifier of the concept, in the same rectangle between parenthesis. Data values are called *value types* and are graphically represented by a dashed, named soft rectangle. Figure 2.1 shows an example of each of these figures. The *population* of an object type are all the instances of the given type.

The relations in ORM2 are called predicates and are modelled based on roles. For each n-ary predicate there are n roles. Roles in ORM2 are represented with boxes and predicates as sequences of one or more of such boxes. Each predicate should include a predicate reading, where the default reading order is left-right

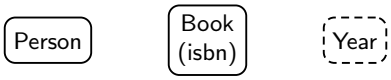


Figure 2.1: An entity type, an entity type with reference mode, and a value type.

or top-down. Arrow-tips show the reversal of the default reading order. For predicates with three or more roles, *place holders* indicate where each role is located. Figure 2.2 shows examples of different predicates.

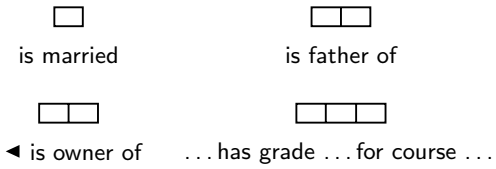


Figure 2.2: Unary, binary, and ternary predicates.

For each role in a predicate there is with one object attached to it, which plays a specific role in the relationship. Role names may be explicitly shown in square brackets. For example, in the model for the binary relationship "is father of", there are two roles: that of parent and the one of child, both of which are played by the same object type, Person. Figure 2.3 shows such a diagram with explicit role names. A possible reading for such a diagram could be "A Person, the parent, *is father of* a Person, the child." The names of the roles may be omitted.

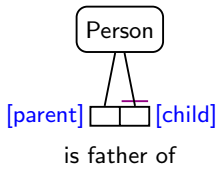


Figure 2.3: Binary predicate with explicit roles.

The association of an n-ary predicate with its n related object types form a *fact type*. In the graphical notation, the association is made with lines from the object type to the roles in the predicate. Figure 2.3 represents a binary predicate where each role, [parent] and [child], is related with the object type *Person*. The *population* of a role are all the instances which participate in the given role.

To model the domain of interest more accurately ORM2 has constructs to refine the model. Object types and roles may be restricted depending on their value or their number of instances. Also the population of each role can be delimited with respect to the frequency of instances in its population. Comparisons between the populations of the roles are provided as well. ORM2 also supports the subtyping of entity types. Lastly, fact types can be objectified and be treated as object types. In the next sections we will introduce the ORM2 constructs used in each of these cases.

2.1.2 Object type constraints

2.1.2.1 Value constraint

Object types can be restricted by the value they can take. This constraint is added to an ORM2 diagram via the enumeration of the possible values in brackets. Ranges can be specified with two dots (..) between the upper and lower bounds, at least one should be specified. The notation also allows the combination of enumerations and ranges. Figure 2.4 shows some examples of the use of this restriction.

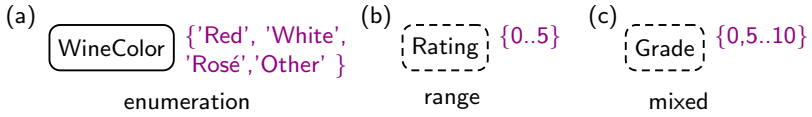


Figure 2.4: ORM2 object types with value constraints.

2.1.2.2 Cardinality constraint

Cardinality constraints restrict the total population of an object type. They can specify an exact number or a range via comparison operators. In the example of Figure 2.5, the number of *Continent*s is restricted to be five.

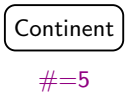


Figure 2.5: An ORM2 object type with a cardinality constraint.

2.1.3 Internal role constraints

Roles can have constraints as well. The role constraints can be divided into two groups, internal and external. The internal constraints restrict the roles in one predicate, whereas the external constraints restrict the roles related to roles in other predicates. These restrictions can span one or more roles, or a join of them (Section 2.1.9).

2.1.3.1 Role value constraint

Even though object types can have value constraints, sometimes is also necessary to limit the values which a role can take, as in Fig. 2.6. Here the values accepted values for the *Month* in the role is open are just *Jan, Feb, Nov* and *Dec*. Similar as the object value constraint, the *role value constraint* can be specified using enumerations, ranges or a mixture of both.

2.1.3.2 Role cardinality constraint

Roles can also be restricted by cardinality constraints. In this case the total population of the roles is restricted. In Fig. 2.7 the ORM2 model the total population for the role is the winner should be at most 1, i.e., there is at most one *Participant* which is the winner.

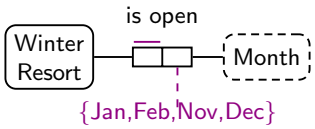


Figure 2.6: An ORM2 diagram with a role value constraint.

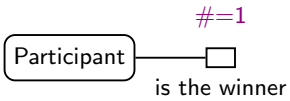


Figure 2.7: An ORM2 diagram with a role cardinality constraint.

2.1.3.3 Internal uniqueness constraint

The *internal uniqueness constraint*, or internal UC for short, sets an upper bound of one for the number of participations of an object type in a role. This means, a role with this restriction accepts only one participation per instance. The internal UC is graphically specified in the diagram with a bar over the restricted role. In Figure 2.8 the restriction asserts that each *Film* is released on at most one *Year*.

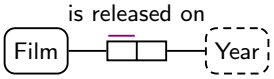


Figure 2.8: An ORM2 diagram with an internal UC.

For predicates with more than one role, there are many possibilities to assign the UCs. As a rule, each n-ary predicate should have at least one UC spanning $n-1$ of its roles. Figures 2.9a, 2.9b and 2.9c present all the possible assignments of UC for a binary predicate. Figures 2.9d shows a UC for a ternary predicate. In this last example the constraint conveys that each combination of the first and third roles should be unique in the population of the predicate.

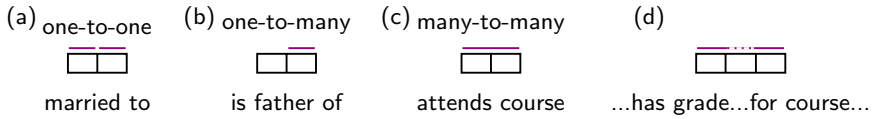


Figure 2.9: Predicates with different kinds of internal UC.

2.1.3.4 Mandatory role constraint

Mandatory role constraints state that all the population of the object type associated to the role should participate in this role. It is depicted by a full dot on either edge of the line joining the object type with the predicate. The fact type in Figure 2.10 has a mandatory constraint for the first role of the predicate is directed by, which means that all *Film*s should be directed by some *Person*.

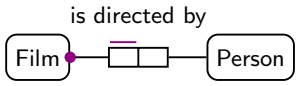


Figure 2.10: An ORM2 diagram with a mandatory role constraint.

2.1.3.5 Internal frequency constraint

A generalisation for the number of times a given instance can participate in a role is given by the *frequency constraint*. The instance participation frequency can be specified with a number, a range or with the use of the comparison operators. In Figure 2.11 each combination of *Team*, *Match* may appear once in the population of the predicate plays on; however, the frequency constraint further specifies that each *Match* can only appear twice.

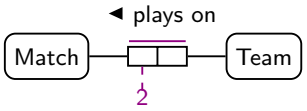


Figure 2.11: An ORM2 diagram with an internal frequency constraint.

2.1.3.6 Preferred internal UC

As we mentioned on Section 2.1.1, object types in ORM2 may have a reference scheme to identify the concept. This is just a syntax shortcut for a mandatory one-to-one relation, as shown in Fig. 2.12. A double bar is drawn to indicate that this value is the reference mode for the related entity.

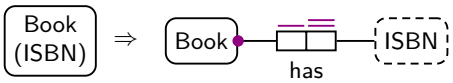


Figure 2.12: Equivalence between reference scheme and the preferred internal UC.

2.1.4 External role constraint

Most of the external role constraints are represented with a dashed line which connects the restricted roles and the specific symbol for the constraint in a circle on this line. The external role constraints are not limited to single roles and most of them can span several roles with possible joins (Section 2.1.9).

2.1.4.1 Value-comparison constraint

This constraint restricts the *value* a role can have in relation to other roles using comparisons operators. The possible comparisons operators are $\leq$, $<$, $>$, and $\geq$. The dashed line ends in a filled arrow tip, to help the reading, and on the circle the comparison operator is located. The circle also has two full dots on the left and right side. Fig. 2.13 shows an example of such a constraint, where the *Year* on which a *Person* died on should be greater than or equal to the *Year* on which the same *Person* was born on.

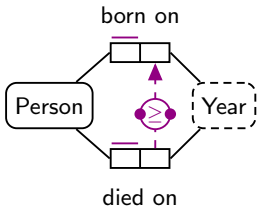


Figure 2.13: An ORM2 diagram with a value comparison constraint.

2.1.5 External set-comparison constraints

Set-comparison constraints restrict the population of one role (or sequences of roles) related to the population of another role.

2.1.5.1 Subset constraint

The *subset constraint* is employed to limit the population of one role as a subset of the population of other role. This constraint also has an arrow tip on the side of the parent role with a subset symbol ($\subseteq$) on the circle. On Figure 2.14 the subset constraint between the *Person* roles in the predicates died on and born on, indicates that any *Person* who died on a *Year*, should have also been born on some *Year*.

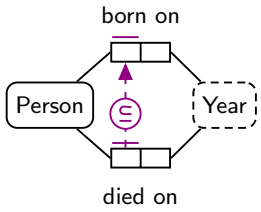


Figure 2.14: An ORM2 diagram with a subset constraint.

2.1.5.2 Equality constraint

The *equality constraint* can be also represented with two subset constraints, one in each direction. It specifies that the population of the roles is the same. The symbol on the circle is an equal sign (=). For Figure 2.15 the population of the roles of has grade for is the same as the population of the roles of took exam of, i.e., all the *Students* which have grade for a *Course* also took the exam of the same *Course*.

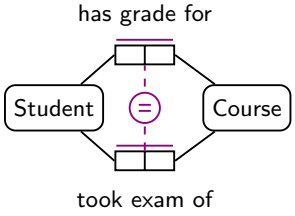


Figure 2.15: An ORM2 diagram with an equality constraint.

2.1.5.3 Exclusion constraint

An *exclusion constraint* prevents that any entity which belongs to the population of one of the restricted roles belongs to the population of any of the other restricted roles. This constraint is represented with a circle with a cross mark inside. Fig. 2.16 show a usage example for this constraint. Here we assert that no *Person* attends the same *Course* he teaches. In this case the constraint spans over both roles of the predicates; therefore, the joint population of both roles is the one used in the constraint.

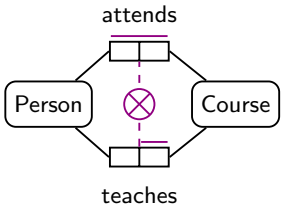


Figure 2.16: An ORM2 diagram with an exclusion constraint.

2.1.5.4 Inclusive-or constraint

For the *inclusive-or constraint*, or external mandatory constraint, all instances of the entities related to the constrained roles should be in the population of at least one of the roles. For Fig. 2.17, this means that all *Student*s should either write a *Thesis* or attend a *Course*. As in the case of the internal mandatory constraint, the symbol for this constraint is a full dot. This constraint does not exclude the possibility of participating in both roles.

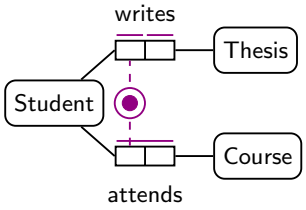


Figure 2.17: An ORM2 diagram with an inclusive-or constraint.

2.1.5.5 Exclusive-or constraint

This constrain is the union of the exclusion constrain with the inclusive-or constraint, and this is reflected also in the symbol which represents it. The instances should belong to the population to at least one of the roles, but not both. With this constraint in Fig. 2.18 all *Student*s should either write a *Thesis* or attend a *Course* but not both.

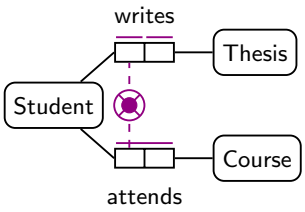


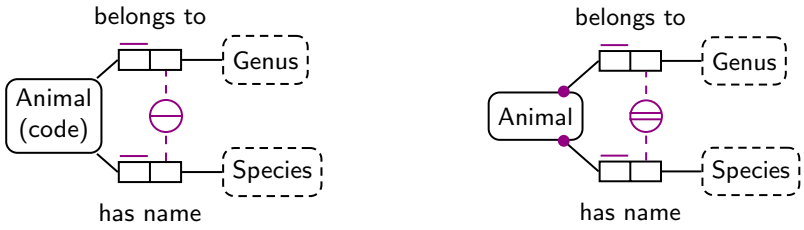
Figure 2.18: An ORM2 diagram with an exclusive-or constraint.

2.1.6 External frequency constraints

2.1.6.1 External uniqueness constraint

The *external uniqueness constraint* limits the number of combinations of instances in the related roles. Each combination should be present just once. The symbol for this constraint is a horizontal line. For example, in Figure 2.19a each combination of *Genus* and *Species* is allowed just once.

It may be the case that this unique combination is selected as the *preferred identifier*, as in Figure 2.19b. For the preferred external UC, we use as symbol a double horizontal line, both of which should touch the circle on both ends to differentiate it from the equal constraint symbol.



(a) External UC. (b) Preferred external UC.

Figure 2.19: ORM2 diagrams with external UC.

2.1.6.2 External frequency constraint

The *external frequency constraint*, as its internal counterpart, is a generalization of the external uniqueness constraint. This constraint assigns the upper or lower bound for the number of times the objects in the constrained roles can appear. The number and the comparison operator are placed on the constraint symbol circle. On the Fig. 2.20 the constraint sets an upper bound to the number of times a *Customer* can request the *Refund* concerns the same *Product*, maximum ten times. In other words, each pair of *Customer*, *Product* is related to at most ten *Refunds*.

2.1.7 Ring constraints

ORM2 also allows the construction of fact types which relate to the same object. To restrict this kind of relations, there are ten *ring constraints* which can also be combined to form?? more?? meaningful constraints. The basic ring constraints

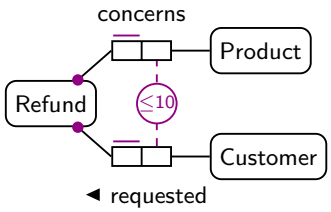


Figure 2.20: An ORM2 diagram with an external frequency constraint.

are: irreflexive, reflexive, asymmetric, symmetric, antisymmetric, intransitive, transitive, strongly intransitive, acyclic, and purely reflexive. The symbols for these constraints are shown in Figure 2.21, together with the symbol for the constraint for an acyclic intransitive ring. As example, Fig. 2.22 shows three

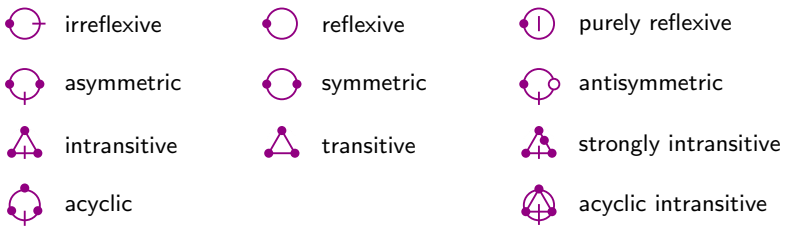
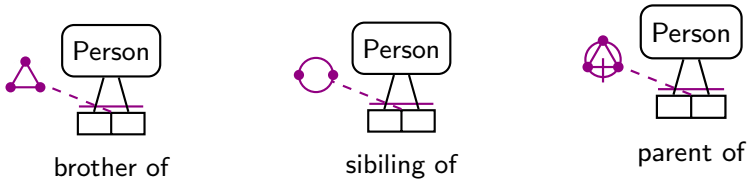


Figure 2.21: All possible ring constraints.

ring predicates, brother of, sibling of, and parent of. In Figure 2.22a, the relation brother of is transitive, because if *Person A* is brother of *Person B*, and *B* is brother of *Person C*, *A* is brother of *C*. But it is not symmetric because it is gender specific. The relation of Figure 2.22b, sibling of, is indeed symmetric. Lastly, for the relation in Figure 2.22c parent of, we specify it is an acyclic intransitive relation as no *Person* can be parent of the parent of his own parent.



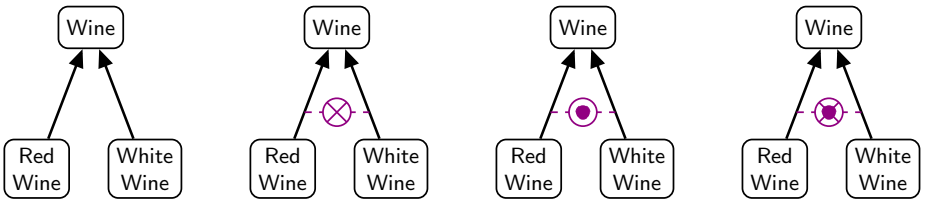
(a) transitive (b) symmetric (c) acyclic intransitive

Figure 2.22: ORM2 diagrams with ring constraints.

2.1.8 Subtyping

ORM2 also provides features to extend objects via *subtyping*. Subtyping is specified with a black arrow with a full tip from the subtype to the parent type. There are also constraints which can be added to the subtypes to specify how their population relate to each other.

Figure 2.23a shows a simple subtyping example, where it is just asserted that *Red Wine* and *White Wine* are subtypes of *Wine*. Fig. 2.23b further specifies that both subtypes are *exclusive*, i.e., an instance cannot belong to both subtypes, in the example no *Red Wine* can be a *White Wine*. On the other hand, Figure 2.23c, specifies that the subtypes are *total* so any instance of *Wine* should belong to any of its subtypes. Lastly, Fig. 2.23d shows how to specify a *partition*, where all the instances of the parent type should belong to one and only one of its subtypes. In our example, all *Wine*s should be *Red Wine*s or *White Wine*s and no *Wine* is both a *Red Wine* and a *White Wine*.



(a) subtyping (b) exclusive (c) total (d) partition

Figure 2.23: ORM2 diagrams with subtyping and subtype constraints.

2.1.9 Join constraints

Several of the external constraints allow for *joins of roles* as arguments. The roles we wish to join should belong to fact types with compatible object types. An example of a join subset constraint is shown in Figure 2.24. We join the populations of the roles in the predicates is created in and is from region so that we obtain all pairs *Country*, *Region* for which there is a *Wine* which is created in the *Country* and the same *Wine* is from the *Region*. The obtained population is then used in the subset constraint. In simpler terms, the subset constraints guarantees that for any *Wine* which is created in a *Country* and also is from a *Region*, this *Region* is located in the same *Country*.

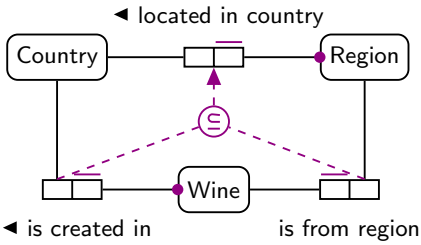


Figure 2.24: An ORM2 diagram with a join subset constraint.

2.1.10 Objectification

Objectification allows us to add relations to predicates by converting them into objects. In an ORM2 diagram this conversion is shown by surrounding the *objectified* predicate with a soft rectangle, as entity types, and assigning it a name. In Figure 2.25 the predicate reviewed has been objectified as *Review* to add the fact has rating to it.

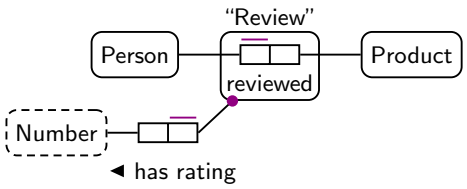


Figure 2.25: An ORM2 diagram with objectification.

In this introduction to the ORM2 graphical notation we omitted some of its components as derived and semiderived fact types, deontic ring constraints and textual constraints.

2.2 Constraints definitions

We first introduce some concepts, as defined by David et al. [4]. In a relational database, a *relation* $r(A_1, \ldots, A_n)$, is formed by r a relation symbol and $\{A_1, \ldots, A_n\}$ a set of pairwise distinct *attributes*. The *arity* of the relation is determined by the number of its attributes, n. A *relational schema* $\mathcal{R}$ is an alphabet of relation symbols.

Two pairwise disjoint sets Γ and Γ_O are assumed, where Γ holds the elements used to identify the usual values of a database and Γ_O holds the elements used to identify *objects*. Each element of Γ and Γ_O is uniquely determined by a constant symbol. The union of both sets compose the *database domain*. Given a relational schema $\mathcal{R}$ a *database instance* or database $\mathcal{D}$ for $\mathcal{R}$ is defined as a set of assertions?? of the form $r(t)$, where r is a relation in $\mathcal{R}$ of arity n and t in a n-tuple of constants over $\Gamma \cup \Gamma_O \cup \texttt{null}$, where `null` is a special constant not in Γ nor Γ_O.

Let $r^{\mathcal{D}} = \{t \mid r(t) \in \mathcal{D}\}$. Given an attribute set $\mathbf{X}$ of $r \in \mathcal{R}$, a database $\mathcal{D}$ for $\mathcal{R}$, and a tuple $t \in r^{\mathcal{D}}$, $t[\mathbf{X}]$ is used to denote the *projection* of t on $\mathbf{X}$. A projection $t[\mathbf{X}]$ is called with no null values if for every $A \in \mathbf{X}$ it holds that $t[A] \neq \texttt{null}$.

A *raw database schema* $\mathcal{DB} = \langle \mathcal{R}, \Sigma \rangle$ is any data source structured as a set of attributes, with no knowledge of the relationships to which they may belong. It consist of $\mathcal{R}$ a relational schema and Σ a set of constraints. The relational schema $\mathcal{R}$, in this case, consists of a unique relation $\mathcal{U}$ which has as attributes all the attributes of the database schema, called the *universal relation*.

The constraints in Σ are of two types, integrity constraints and null value constraints. Integrity constraints restrict which tuples can be part of a relation [7]. Null value constraints deal with the `null` values in the attributes.

2.2.1 Integrity constraints

A *functional dependency* (FD) of the form $r : \mathbf{X} \rightarrow A$ holds in a relation r of a database $\mathcal{D}$ if for every tuple in r the value of the attribute A depends on the value of the attributes in $\mathbf{X}$. In other words, for each set of values that is assigned to $\mathbf{X}$ corresponds only one value for A. For example, in the relation the relation $attendance(Course, Student, Professor)$ if the FD

$attendance : \{Course\} \rightarrow Professor$ holds then for each $Course$ there is only one $Professor$. The FDs further allow us to define the *key* of a relation. A subset of attributes of r $\mathbf{X}$ is a key or candidate key of r, $key(r, \mathbf{X})$, if for every attribute B of r such that $B \notin \mathbf{X}$ it holds that $r : \mathbf{X} \rightarrow B$ and no proper subset of $\mathbf{X}$ has this property.

A *multivalued dependency* (MVD) is an assertion of the form $r : \mathbf{X} \twoheadrightarrow \mathbf{Y}$. The MVD are a generalization of the FDs [7]. For this constraint each set of values assigned to $\mathbf{X}$ may correspond to more than one set of values of $\subset Y$. However, the set of the possible values assigned to the set of attributes $\mathbf{Y}$ depends on the set of values assigned to $\mathbf{X}$. For example, given the relation $attendance(Course, Student, Professor)$, if there is a MVD between $Course$ and $Student$ it would mean that the set of $Students$ which attend a $Course$ depends only on the $Course$.

An *inclusion dependency* (IND) of the form $r_1[\mathbf{X}] \subseteq r_2[\mathbf{Y}]$ constraints the values of a relation with respect to the values of other relation. That is, for each tuple in r_1 the values assigned to the attributes in $\mathbf{X}$ should also be assigned to the attributes in $\mathbf{Y}$ of some tuple in the relation r_2. If in an IND $key(r_2, \mathbf{Y})$ also holds, $\mathbf{X}$ is called a foreign key of r_1. For instance, a IND between the relations $student(StudentId)$ and $masterStudent(StudentId, MastersName)$, $masterStudent[\{StudentId\}] \subseteq student[\{StudentId\}]$ means that all $masterStudents$s should be $students$.

The last integrity constraint is the *exclusion dependency* (ED) $r_1[\mathbf{X}] \cap r_2[\mathbf{Y}] = \emptyset$. As the INDs, this constraint restricts the values of a relation with respect to the values of other relation. However, instead of requiring the values in the other relation to be present, this constraint requires that the values are *not* present. In other words, every set of values assigned to the attributes of $\mathbf{X}$ in a tuple of r_1 should not be assigned to the attributes in $\mathbf{Y}$ of r_2. For example, an ED between the relations $employee(PersonId)$ and $underage(PersonId)$ insures that no $underage$ person is an $employee$.

2.2.2 Null value constraints

The last three constraints allow us to explicitly define how the `null`s for each attribute should be handled. In a relational model, the `null` value allows for two interpretations [36], *unknown* or *nonexistent*. Given a tuple t and an attribute A, if $t[A] =$ `null` under an unknown interpretation this `null` represents some value in Γ which is missing from the tuple in the database. On the other hand, under a nonexistent interpretation, the value for $t[A]$ does not exist.

For example, in the relation $person(Name, MiddleName, FatherName)$, the `null` values in attribute $FatherName$ are interpreted as unknown because an actual value exists. On the other hand, if the attribute $MiddleName$ is `null` for some instance, it means that the middle name does not exist, i.e., this person has no middle name.

A *not null constraint* for an attribute A in a relation r prevents the appearance of `null` values in attribute A for all tuples in r. For example, in the relation $course(CourseId, Name)$, to enforce that $Name$ should not be null we could add the not null constraint $not\text{-}null(course, Name)$.

A *mandatory participation constraint* (MPC) for an attribute A in a relation r indicates that for all tuples in r, the attribute A has a value. Therefore, if for a tuple A is `null`, this null represents an unknown value. As in the case of the relation $person(Name, MiddleName, FatherName)$, a constraint $exists(person, FatherName)$ should be added to indicate that the $FatherName$ always exists; although, it may be unknown.

A *MPC w.r.t. the left-hand side of an FD* is a special case of MPC. Given an attribute A in a relation r and a FD $r : \mathbf{X} \rightarrow A$, this constraint states that if in a tuple the attributes in $\mathbf{X}$ are not null, then the value for A in the tuple should also exists. That is, in any tuple of r any `null` value in A, if $\mathbf{X}$ is not null, is an unknown value. For instance in the relation $livesIn(PersonId, City, Country)$ with a FD $livesIn : City \rightarrow Country$ means that if we know in which $City$ the person *lives in*, then we should also know in which $Country$, i. e., if $City$ is not `null` then $Country$ is not `null` as well.

In Table 2.1 we present a formal definition of all these constraints.

In the following, r is a relation in $\mathcal{R}$, A and B are attributes of r, and $\mathbf{X}$ and $\mathbf{Y}$ are subsets of attributes of r. For the integrity constraints only projections with no null values are considered.

Constraint	is satisfied in a database $\mathcal{D}$ iff
Functional dependency $r : \mathbf{X} \rightarrow A$	for each $t_1, t_2 \in r^{\mathcal{D}}$ if $t_1[\mathbf{X}] = t_2[\mathbf{X}]$ then $t_1[A] = t_2[A]$
Multivalued dependency $r : \mathbf{X} \twoheadrightarrow \mathbf{Y}$	for each $t_1, t_2 \in r^{\mathcal{D}}$ if $t_1[\mathbf{X}] = t_2[\mathbf{X}]$ then there exist $t_3, t_4 \in r^{\mathcal{D}}$ s.t. (i) $t_3[\mathbf{X}] = t_1[\mathbf{X}]$, $t_3[\mathbf{Y}] = t_1[\mathbf{Y}]$, and $t_3[\mathbf{Z}] = t_2[\mathbf{Z}]$ (ii) $t_4[\mathbf{X}] = t_1[\mathbf{X}]$, $t_4[\mathbf{Y}] = t_2[\mathbf{Y}]$, and $t_4[\mathbf{Z}] = t_1[\mathbf{Z}]$ for $\mathbf{Z}$ set of all attributes of r that are neither in $\mathbf{X}$ nor in $\mathbf{Y}$
Inclusion dependency $r_1[\mathbf{X}] \subseteq r_2[\mathbf{Y}]$	for each $t_1 \in r_1^{\mathcal{D}}$ there exists $t_2 \in r_2^{\mathcal{D}}$ s.t. $t_1[\mathbf{X}] = t_2[\mathbf{Y}]$ where r_1, r_2 are relations in $\mathcal{R}$ and $\lvert\mathbf{X}\rvert = \lvert\mathbf{Y}\rvert$
Exclusion dependency $r_1[\mathbf{X}] \cap r_2[\mathbf{Y}] = \emptyset$	there are no two tuples $t_1 \in r_1^{\mathcal{D}}$ and $t_2 \in r_2^{\mathcal{D}}$ s.t. $t_1[\mathbf{X}] = t_2[\mathbf{Y}]$ where r_1, r_2 are relations in $\mathcal{R}$ and $\lvert\mathbf{X}\rvert = \lvert\mathbf{Y}\rvert$
Not null constraint *not-null*(r, A)	for each $t \in r^{\mathcal{D}}$ it holds that $t[A] \neq \texttt{null}$
Mandatory participation constraint *exists*(r, A)	for each $t \in r^{\mathcal{D}}$ such that for each $t[A] = \texttt{null}$, this `null` denotes some value $v \in \Gamma$ for $t[A]$
MPC w.r.t. the left-hand side (LHS) of an FD *exists*$(r, A)_{LHS(\mathbf{X} \rightarrow A)}$	for each $t \in r^{\mathcal{D}}$ s.t. $t[B] \neq \texttt{null}$ for all $B \in \mathbf{X}$, each `null` where $t[A] = \texttt{null}$ denotes some value $v \in \Gamma$ for $t[A]$ where $r : \mathbf{X} \rightarrow A$ is an FD

Table 2.1: Integrity and `null` value constraints

Property	Abbrev.
Appellation_Id	AID
Appellation_Name	AN
ProductAttribute_Id	PAID
ProductAttribute_ImageUrl	PAIU
ProductAttribute_Name	PAN
Producer_GeoLocation_Url	PG
Producer_Id	PID
Producer_ImageUrl	PIU
Producer_Name	PN
Producer_Url	PU
Region_Id	RID
Region_Name	RN
WineType_Id	TID

Property	Abbrev.
WineType_Name	TN
VarietalBlend_Id	VBID
VarietalBlend_Name	VBN
GeoLocation_Url	WPG
WineProduct_Id	WPID
PriceMin	WPM
Name	WPN
PriceRetail	WPR
Ratings_HighestScore	WPT
Url	WPU
Reviews_HighestScore	WPV
PriceMax	WPX
VintageYear	WPY

Table 2.2: Property names and abbreviations

2.3 Data preprocessing

The data provided by Wine.com through their public API (api.wine.com) is available in XML or JSON formats. The data used was obtained via the Catalog API. The structure of data is a list of *objects* which may contain *properties* with data values and other objects [38].

We changed the names of some of the properties and objects to remove ambiguous terms. For example, we changed Year to VintageYear and *Varietal* to *VarietalBlend*. The object *Vineyard* actually refers to the wine producer, and we changed the name to *Producer*. Finally, we cleaned the data by removing those entries with erroneous information.

The list of properties obtained from the Catalog and their description are shown in Appendix A. Some properties, like the URLs to other products were ignored for the construction of the conceptual schema.

The used properties with their abbreviations are shown in Table 2.2.

In Figure 2.26 we show a sample of anonymized data obtained from the XML. We will use this data sample to illustrate the methodology followed in the next sections.

2.4 Data constraints

The adopted methodology converts a raw database schema into a conceptual schema [4]. Therefore, first we obtained such a database schema for our data. The relational schema $\mathcal{R}$, as noted before, consist of a universal relation, denoted by $\mathcal{U}$ which for our case included all the attributes defined in Table 2.2. Contrary to what David et al. [4] assume, we lacked the logical schema of the relational data. Therefore, we extracted the constraints using data mining techniques based on the XML structure of the data [38], and in the actual data obtained. We deduced 29 constraints for this data source.

Constraints (2.1) to (2.7) are functional dependency constraints and are described in Subsection 2.4.1. We found only one multivalued dependency

t_i	AID	AN	PAID	PAIU	PAN	PG	PID	PIU	PN
t_1	4	Provence				.../wnmap=1	1	w1.jpg	La Bastide Blanche
t_2	1	Veneto					3	w3.jpg	Lamberti
t_3	2	Piedmont	2		Bold		2		Cascina Adelaide

t_i	PU	RID	RN	TID	TN	VBID	VBN	WPG
t_1	.../w=1	2	France	3	Rosé Wine		Rosé	.../map=1
t_2	.../w=3	1	Italy	1	Sparkling	3	Rosé	
t_3	.../w=2	1	Italy	2	Red Wines	4	Nebbiolo	

t_i	WPID	WPM	WPN	WPR	WPT	WPU	WPV	WPX	WPY
t_1	1	2.8	La BBB Rosé 2004	2.8	90	.../detail=1	5	6.7	2004
t_2	2	3.3	Lamberti Rose Spumante	5.8	0	.../detail=2	0	7.3	NV
t_3	3	9.6	C. Adelaide (1.5 L) 2010	9.6	97	.../detail=3	0	9.6	2010

Figure 2.26: Sample data

constraint (2.8), shown in Subsection 2.4.2. Because we started with a universal relation $\mathcal{U}$, there were no inclusion nor exclusion dependency constraints.

The `null` value constraints are shown in the last three subsections of this section. The not null constraints [(2.9) to (2.16)] in Subsection 2.4.3, the mandatory participation constraints [(2.17) to (2.20)] in Subsection 2.4.4, and the MPC w.r.t. the LHS [(2.24) to (2.21)] in Subsection 2.4.5. Also Table 2.3 shows the semantics of the `null` value for each attribute.

Integrity constraints

In the next subsections we use the abbreviations introduced in Table 2.2. For both the FDs and the MVDs we omit the relation which is always $\mathcal{U}$. We merge those FDs with the same LHS into a single FD to avoid repetition.

2.4.1 Functional dependencies

A wine product has a product Name and at most one VintageYear. It also has a detail Url, a PriceMin, a PriceMax, and a PriceRetail. It may have at most one GeoLocation_Url as location URL. It has a Ratings_HighestScore and a community Reviews_HighestScore. The product also corresponds to one appellation, and is made by one producer created with grapes of one varietal/blend.

$$\{WPID\} \rightarrow \{WPN, WPY, WPU, WPM, WPX, WPR, WPG, WPT, WPV, AID, PID, VBID\} \tag{2.1}$$

An appellation has an Appellation_Name, and corresponds to one region.

$$\{AID\} \rightarrow \{AN, RID\} \tag{2.2}$$

A region has one Region_Name.

$$\{RID\} \rightarrow \{RN\} \tag{2.3}$$

Attr.	Null type
WPID	not-null
WPM	not-null
WPN	not-null
WPR	not-null
WPT	not-null
WPU	not-null
WPV	not-null
WPX	not-null
AID	unknown
AN	unknown
PID	unknown
PN	unknown
PU	unknown

Attr.	Null type
RID	unknown
RN	unknown
TID	unknown
TN	unknown
VBID	unknown
VBN	unknown
WPY	unknown
PAID	nonexistent
PAIU	nonexistent
PAN[1]	nonexistent
PG	nonexistent
PIU	nonexistent
WPG	nonexistent

Table 2.3: Null types by attribute

A varietal/blend has one VarietalBlend_Name and corresponds to a wine type.

$$\{VBID\} \to \{VBN, TID\} \tag{2.4}$$

A wine type has a WineType_Name.

$$\{TID\} \to \{TN\} \tag{2.5}$$

A producer has a Producer_Name and a Producer_Url as description URL. It may also have at most one Producer_ImageUrl and a location shown in at most one Producer_GeoLocation_Url.

$$\{PID\} \to \{PN, PU, PIU, PG\} \tag{2.6}$$

A product attribute has a ProductAttribute_Name and has at most one Product-Attribute_ImageUrl.

$$\{PAID\} \to \{PAN, PAIU\} \tag{2.7}$$

2.4.2 Multivalued dependencies

A wine product may have several product attributes

$$WPID \twoheadrightarrow \{PAID\} \tag{2.8}$$

Null constraints

Each attribute of $\mathcal{U}$ has three possibilities regarding the null values. It may be never `null`, it may be `null` and unknown, or `null` and nonexistent. Table 2.3 shows which possibility corresponds to each attribute. Based on this table and in the integrity constraints we obtained the following constraints.

2.4.3 Not null constraints

[1] Depends on the value of PAID. See constraints (2.7) and (2.23)

$$\textit{not-null}(\mathcal{U},WPID) \quad (2.9) \qquad \textit{not-null}(\mathcal{U},WPT) \quad (2.13)$$
$$\textit{not-null}(\mathcal{U},WPN) \quad (2.10) \qquad \textit{not-null}(\mathcal{U},WPU) \quad (2.14)$$
$$\textit{not-null}(\mathcal{U},WPM) \quad (2.11) \qquad \textit{not-null}(\mathcal{U},WPV) \quad (2.15)$$
$$\textit{not-null}(\mathcal{U},WPR) \quad (2.12) \qquad \textit{not-null}(\mathcal{U},WPX) \quad (2.16)$$

2.4.4 Mandatory participation constraints

$$\textit{exists}(\mathcal{U},AID) \quad (2.17) \qquad \textit{exists}(\mathcal{U},VBID) \quad (2.19)$$
$$\textit{exists}(\mathcal{U},PID) \quad (2.18) \qquad \textit{exists}(\mathcal{U},WPY) \quad (2.20)$$

2.4.5 MPC w.r.t. left-hand side

$$\textit{exists}(\mathcal{U},VBN)_{LHS(VBID \to VBN)} \quad (2.21)$$
$$\textit{exists}(\mathcal{U},TID)_{LHS(VBID \to TID)} \quad (2.22)$$
$$\textit{exists}(\mathcal{U},PAN)_{LHS(PAID \to PAN)} \quad (2.23)$$

$$\textit{exists}(\mathcal{U},AN)_{LHS(AID \to AN)} \quad (2.24) \qquad \textit{exists}(\mathcal{U},PN)_{LHS(PID \to PN)} \quad (2.27)$$
$$\textit{exists}(\mathcal{U},RID)_{LHS(AID \to RID)} \quad (2.25) \qquad \textit{exists}(\mathcal{U},PU)_{LHS(PID \to PU)} \quad (2.28)$$
$$\textit{exists}(\mathcal{U},RN)_{LHS(RID \to RN)} \quad (2.26) \qquad \textit{exists}(\mathcal{U},TN)_{LHS(TID \to TN)} \quad (2.29)$$

2.5 Developing the conceptual schema

David et al. [4] methodology consists of five phases which are labelling existent nulls, normalization to the 4NF, processing nonexistent nulls, introduction of object identifiers and other constraints, and finally the generation conceptual schema.

The main idea of this methodology is to translate the data constraints into conceptual constraints. Using the functional and multivalued dependencies the initial $\mathcal{U}$ is decomposed into several relations which fulfil the restrictions. To guarantee a lossless decomposition, the methodology performs an analysis of the meaning of the null values before the actual decomposition. Attributes with unknown null values are assigned a new value, which should satisfy the integrity constraints.

On each of the following subsections, we will briefly explain each phase and how we applied it to our case of study. We will continue to use the abbreviations introduced in Table 2.2.

2.5.1 Labelling existent nulls

In this phase the attributes for which `null` represents an unknown value are given a filler value. These filler values should adhere to the integrity constraints of the database schema defined in Σ [27]. Once all the unknown `null` values are assigned a new value, all the MPC are converted into no null value constraints. From this point on all the `null` values are nonexistent.

For our study case, we defined these integrity constraints in Section 2.4. Therefore, we assign the filler values such that for any attribute A, which can

t_i	AID	AN	PAID	PAIU	PAN	PG	PID	PIU	PN
t_1	4	Provence				.../wnmap=1	1	w1.jpg	La Bastide Blanche
t_2	1	Veneto					3	w3.jpg	Lamberti
t_3	2	Piedmont	2		Bold		2		Cascina Adelaide

t_i	PU	RID	RN	TID	TN	VBID	VBN	WPG
t_1	.../w=1	2	France	3	Rosé Wine	*100*	Rosé	.../map=1
t_2	.../w=3	1	Italy	1	Sparkling	3	Rosé	
t_3	.../w=2	1	Italy	2	Red Wines	4	Nebbiolo	

t_i	WPID	WPM	WPN	WPR	WPT	WPU	WPV	WPX	WPY
t_1	1	2.8	La BBB Rosé 2004	2.8	90	.../detail=1	5	6.7	2004
t_2	2	3.3	Lamberti Rose Spumante	5.8	0	.../detail=2	0	7.3	NV
t_3	3	9.6	C. Adelaide (1.5 L) 2010	9.6	97	.../detail=3	0	9.6	2010

Figure 2.27: Sample data with no unknown `null`s

be `null`, and any FD $\mathbf{X} \to A$ where A appears in the RHS, if $t_1[\mathbf{X}] = t_2[\mathbf{X}]$ for any $t_1, t_2 \in \mathcal{U}$, then it holds that $t_1[A] = t_2[A] = v$, v a filler value. A special case resulted from constraint (2.7), because the LHS may also be `null`, we added a further restriction so that, if $t_1[PAID] = t_2[PAID] \neq$ `null` then it should be the case that $t_1[PAN] = t_2[PAN]$. We also updated all the MPC in Subsection 2.4.4 to not null constraints.

For the sample on Figure 2.26, the VBID for tuple t_1 is `null`. However, this `null` corresponds to an *unknown* value. As this attribute does not appear in the RHS of any FD we assign any value to it . The resulting relation can be seen in Figure 2.27.

2.5.2 Normalization to the 4NF

Normalization is a process proposed by Codd [3] by which the relations in a database are organized to eliminate redundancy and support data integrity. There are five Normal Forms (NF) and on each NF the relations of the database schema must fulfil with certain restrictions.

For the relations of a database schema to be in First Normal Form (1NF) there must not be duplicate tuples in any relation and all the attributes must contain simple values [2]. The Second Normal Form (2NF) deals with the functional dependencies on relation keys which are not atomic. A relation is in 2NF iff it is in 1NF and for all $r \in \mathcal{R}$ if $key(r, \mathbf{X})$ and $|\mathbf{X}| \geq 2$ then there is no FD $r : \mathbf{Y} \to B$ such that $B \notin \mathbf{X}$ and $\mathbf{Y} \subset \mathbf{X}$. In other words, all the non-key attributes of r functionally depend on the whole set of attributes of its key or candidate keys.

The third Normal Form (3NF) tackles the transitive functional dependencies. For a relation r to be in 3NF it must be in 2NF and any attribute not in a key or candidate key must functionally depend directly on a key of r. That is, if B is an attribute of r that is not in a key of r, then there is no $\mathbf{X}$ and $\mathbf{Y}$ such that $key(r, \mathbf{X})$, not $key(r, \mathbf{Y})$, $r : \mathbf{X} \to \mathbf{Y}$ and $r : \mathbf{Y} \to B$. The Fourth Normal Form (4NF), proposed by Fagin [7], deals with MVD. A relation r is in 4NF iff it is in 3NF and if a nontrivial MVD $r : \mathbf{X} \twoheadrightarrow \mathbf{Y}$ exists, then it is also the case that $r : \mathbf{X} \to A$ for all A attribute of r. In other words, all dependencies in a relation

have in the LHS a key for r. We will not discuss the Fifth Normal Form in this work.

Given the nine FDs and the MVD in Section 2.4 we normalized the universal relation $\mathcal{U}$. For the 1NF the Name attribute was not a simple value as it contained information regarding the VintageYear and a new attribute we called BottleType and will abbreviate as BT. This new attribute was included to the right-hand side (RHS) of constraint (2.1), and because it has a default value, a not null constraint was also added. We then changed the attribute name from Name (WPN) to WineName (WN). For example, the attribute WineName for the tuple t_3 in the sample of Figure 2.27 after the 1NF would be contain the string "C. Adelaide" and the new attribute BottleType will be assigned the string "1.5 L".

Further evaluation of the data allowed us to add the following FDs.

$$\{WN\} \rightarrow \{PID\} \tag{2.30}$$

$$\{WN, WPY\} \rightarrow \{AID, VBID\} \tag{2.31}$$

$$\{WN, WPY, BT\} \rightarrow \{WPID\} \tag{2.32}$$

Constraint (2.30) means that a WineName belongs to exactly one Producer. Constraint (2.31) indicates that a WineName of a given VintageYear belongs to an appellation, and is created with grapes of a varietal/blend. The WineName, the VintageYear and the BottleType determine a wine product as stated by constraint (2.32). At this point, the key for $\mathcal{U}$ was the pair of attributes $WPID$ and $PAID$. According to 2NF all non key attributes should depend on the whole key. This was not the case mainly due to FDs (2.1) and (2.7). Therefore, we decomposed $\mathcal{U}$ into three relations, *wineProduct*, *productAttribute* and *hasAttribute*, where *productAttribute* had all the attributes of constraint (2.7), *hasAttribute* the attributes $WPID$ and $PAID$, and *wineProduct* all the attributes of $\mathcal{U}$ not in *productAttribute*.

The sample data would be divided into the three relations of Figure 2.28.

For the 3NF we analysed the transitive FDs. In *wineProduct* there were several of such cases, given by the composition of FD (2.1) and the constraints from (2.2) to (2.6), (2.30) and (2.31). Finally, as there were no MVD on any relation in the 3NF, this schema was also in the 4NF.

The relations in our schema after normalization were the following

$$wineProduct(\underline{WPID}, \underline{\underline{WN, WPY, BT}}, WPG, WPU, WPM, WPX, WPR, WPT, WPV)$$

$$appellation(\underline{AID}, AN, RID) \qquad region(\underline{RID}, RN)$$

$$hasAttribute(\underline{WPID, PAID}) \qquad varietalBlend(\underline{VBID}, VBN, TID)$$

$$producer(\underline{PID}, PN, PU, PIU, PG) \qquad wine(\underline{WN}, PID)$$

$$productAttribute(\underline{PAID}, PAN, PAIU) \qquad wineType(\underline{TID}, TN)$$

$$yearlyWine(\underline{WN, WPY}, AID, VBID)$$

Keys are underlined and candidate keys are double underlined. The normalized schema for the sample is shown in Figure 2.29. The *wineProduct* relation was divided into eight relations by the FDs. For the sample only the primary keys are shown in bold face.

wineProduct

t_i	AID	AN	PG	PID	PIU	PN	PU	RID
t_1	4	Provence	.../wnmap=1	1	w1.jpg	La Bastide Blanche	.../w=1	2
t_2	1	Veneto		3	w3.jpg	Lamberti	.../w=3	1
t_3	2	Piedmont		2		Cascina Adelaide	.../w=2	1

wineProduct

t_i	RN	TID	TN	VBID	VBN	WPG	WPID	WPM
t_1	France	3	Rosé Wine	100	Rosé	.../map=1	1	2.8
t_2	Italy	1	Sparkling	3	Rosé		2	3.3
t_3	Italy	2	Red Wines	4	Nebbiolo		3	9.6

wineProduct

t_i	WPN	WPR	WPT	WPU	WPV	WPX	WPY
t_1	La BBB Rosé	2.8	90	.../detail=1	5	6.7	2004
t_2	Lamberti Rose Spumante	5.8	0	.../detail=2	0	7.3	NV
t_3	C. Adelaide	9.6	97	.../detail=3	0	9.6	2010

productAttribute

t_i	PAID	PAIU	PAN
t_1			
t_2			
t_3	2		Bold

hasAttribute

t_i	WPID	PAID
t_1	1	
t_2	2	
t_3	3	2

Figure 2.28: Sample data in 2NF

2.5.2.1 Inclusion dependencies

Since David et al. [4] use the *universal instance assumption* (UIA) as defined by Bernstein [2], all the generated relations can be seen as projections of $\mathcal{U}$. Thus, inclusion dependencies exist between the new relations and $\mathcal{U}$. By transitivity of the subset relation in the INDs, equality dependencies between the attributes of the new relations are inferred.
For our case study, we considered only such INDs which involve at least one key

attribute. The inferred INDs are the following:

$$yearlyWine[AID] \subseteq appellation[AID] \tag{2.33}$$
$$hasAttribute[PAID] \subseteq productAttribute[PAID] \tag{2.34}$$
$$wine[PID] \subseteq producer[PID] \tag{2.35}$$
$$appellation[RID] \subseteq region[RID] \tag{2.36}$$
$$varietalBlend[TID] \subseteq wineType[TID] \tag{2.37}$$
$$yearlyWine[VBID] \subseteq varietalBlend[VBID] \tag{2.38}$$
$$wineProduct[WN] \subseteq wine[WN] \tag{2.39}$$
$$yearlyWine[WN] \subseteq wine[WN] \tag{2.40}$$
$$hasAttribute[WPID] \subseteq wineProduct[WPID] \tag{2.41}$$
$$wineProduct[WN, WPY] \subseteq yearlyWine[WN, WPY] \tag{2.42}$$

$$appellation[AID] \subseteq yearlyWine[AID] \tag{2.43}$$
$$producer[PID] \subseteq wine[PID] \tag{2.44}$$
$$region[RID] \subseteq appellation[RID] \tag{2.45}$$
$$wineType[TID] \subseteq varietalBlend[TID] \tag{2.46}$$
$$varietalBlend[VBID] \subseteq yearlyWine[VBID] \tag{2.47}$$
$$wine[WN] \subseteq wineProduct[WN] \tag{2.48}$$
$$wine[WN] \subseteq yearlyWine[WN] \tag{2.49}$$
$$yearlyWine[WN, WPY] \subseteq wineProduct[WN, WPY] \tag{2.50}$$
$$productAttribute[PAID] \subseteq hasAttribute[PAID] \tag{2.51}$$
$$wineProduct[WPID] \subseteq hasAttribute[WPID] \tag{2.52}$$

All the new INDs are added to Σ' of the normalized schema $\texttt{norm}(\mathcal{DB}) = \langle \mathcal{R}', \Sigma' \rangle$. If the set of attributes on the RHS is also a key of the relation in the RHS, then in agreement with the definition introduced in Section 2.4 that set of attributes is a foreign key of the relation on the LHS. This is the case for the dependencies from (2.33) to (2.42).

2.5.3 Processing nonexistent nulls

The INDs just added assume that the projections for each attribute are the same for all relations. However, it is important to remember that some of the attributes may contain nonexistent `null` values. For example, the dependency (2.52) presumes that all *wineProduct*s should also be in the relation *hasAttribute* which is not always true. Therefore, in this phase the `null` values are removed from the relations.

Let r be a relation $r(A_1, \ldots, A_n)$, with key $\mathbf{K}$, where at least one of its attributes can be null. To remove the `null` values from r, it is decomposed into 2^n relations resulting from the projections over the set of attributes of r from the power set of its attributes. Only tuples with no `null` values are included in these relations.

From all these relations only those with all the attributes of $\mathbf{K}$ are considered. For each new relation r' there is new foreign key constraint of the form $r'[\mathbf{K}] \subseteq$

$r[\mathbf{K}]$. If all the attributes of r' did not have `null`s in the original relation r, then it is also the case that $r[\mathbf{K}] \subseteq r'[\mathbf{K}]$.

Given all the foreign key constraints, there are two cases to consider:

1. If the key attributes for a relation r are composed of one foreign key.

2. If the key attributes for a relation r are composed of more than one foreign keys.

For the first case, new foreign constraints between the decomposed relations are generated $r''[\mathbf{K}] \subseteq r'[\mathbf{K}]$, in any case where r'' has attributes which had `null`s and r' has only non nullable attributes with $\mathbf{K}$ the key for both relations. For the second case, any IND $r''[\mathbf{K}] \subseteq r'[\mathbf{K}]$ where the key of r' is composed of more than one foreign key, and r' has nullable attributes, is dropped.

For our case the only attributes which can contain nulls with no constraints are $PAID$, $PAIU$, PIU, PG, and WPG, which correspond to relations *hasAttribute*, *productAttribute*, *producer*, and *wineProduct*. The attribute PAN may contain `null`s only if $PAID$ is `null`. After the decomposition we obtain this new schema:

$$wineProduct(\underline{WPID}, \underline{\underline{WN, WPY, BT}}, WPU, WPM, WPX, WPR, WPT, WPV)$$
$$wineProductWithGeoLocation(\underline{WPID}, WPG)$$
$$productAttributeWithImage(\underline{PAID}, PAIU)$$
$$yearlyWine(\underline{WN, WPY}, AID, VBID)$$

$appellation(\underline{AID}, AN, RID)$
$hasAttribute(\underline{WPID, PAID})$
$producer(\underline{PID}, PN, PU)$
$producerWithImage(\underline{PID}, PIU)$
$producerWithGeoLocation(\underline{PID}, PG)$
$productAttribute(\underline{PAID}, PAN)$
$region(\underline{RID}, RN)$
$varietalBlend(\underline{VBID}, VBN, TID)$
$wine(\underline{WN}, PID)$
$wineType(\underline{TID}, TN)$

We dropped constraints (2.51) and (2.52) and added the following new foreign key constraints.

$$producerWithImage[PID] \subseteq producer[PID] \tag{2.53}$$
$$producerWithGeoLocation[PID] \subseteq producer[PID] \tag{2.54}$$
$$wineProductWithGeoLocation[WPID] \subseteq wineProduct[WPID] \tag{2.55}$$
$$productAttributeWithImage[PAID] \subseteq productAttribute[PAID] \tag{2.56}$$

For our sample data all the nullable attributes contain `null`s. The decomposition to remove all `null` values, added three new relations to the schema. This schema is shown in Figure 2.30. The attribute $PAIU$ is not in the schema any more, as all its values in the sample are `null` and this schema has no `null` values.

2.5.4 Object IDs, other constraints

For the next phase, the conceptual entities are identified and assigned a unique *Object Identifier* to identifying, instead of using its attributes as identifiers. It is suggested to insert a new attribute to act as the key of the relation. Foreign keys should be updated accordingly. These new attributes belong to Γ_O. David et al. [4] give a guide to identify entities, if the key of a relation r is disjoin from its foreign keys, then r is a candidate entity. In this phase other constraints can be added to the constraints, as well as more expressive names to the relations. The resulting database?? schema is denoted by $\mathcal{DB}^* = \langle \mathcal{R}^*, \Sigma^* \rangle$.

In our new schema, most of the relations generated already had an object identifier, except for *yearlyWine* and *wine*. We introduced OIDs for these relations and updated the associated relations and constraints. Our schema with OIDs is shown in Figure 2.31. For the sample, the schema with new OIDs is shown in Figure 2.32. We decided to drop also the dependencies (2.43) to (2.50) which although supported by our data, conceptually were too strong. We wanted to be able to model, for example, appellations from which no wine was made. All the updated constraints for the new schema are shown in Appendix B

2.5.5 Generation the conceptual schema

The last phase of the methodology is the generation of the conceptual schema form the database schema $\mathcal{DB}^*$. To this purpose, the identities and the relationships have to be identified?? from the relations in R^*. This is achieved using the associations of the keys and foreign keys and the constraints in Σ^*. Each relation r with key **K** and foreign key attributes **FK** is classified into one of the following groups:

specific
: if **K** is a single OID and a foreign key of r.

base
: if either **K** is an OID and r is not a specific relation or **K** and **FK** are disjoint.

relationship
: if **K** has only foreign keys and the number of foreign keys is at least two.

multivalued
: if **FK** is a proper subset of **K**

ambiguous
: if none of the previous conditions hold.

Table 2.4 summarizes this classification. For the ambiguous relations, input from the domain expert is required to disambiguate. Each kind of relation is associated to a conceptual schema element. Base relations correspond to entities and specific relations to sub-entities. Relationship relations are conceptual relationships and multivalued relations correspond to multivalued attributes. The attributes of the relations, naturally correspond to attributes of the conceptual elements. Although the generate schema by David et al. [4] is an Entity-Relationship (ER) schema, we generated an ORM2 model. We consider ORM2

$\mathbf{K} \otimes \mathbf{FK}$	$\lvert\mathbf{K}\rvert = 1$	$\lvert\mathbf{K}\rvert > 1$
$\mathbf{FK} = \emptyset$	base	base
$\mathbf{K} \cap \mathbf{FK} = \emptyset$	base	base
$\mathbf{K} \subseteq \mathbf{FK}$	specific	relation
$\mathbf{K} \supset \mathbf{FK}$	-	multivalued
any other case	-	ambiguous

Table 2.4: Classification of relations based on keys.

has a richer syntax for constraints and subtyping than ER, which at the end will provide us with a richer ontology.

The classification of the relations in $\mathcal{R}^*$ for our study is the following:

base	appellation	producer	region	wine	productAttribute
	varietalBlend	wineType	wineProduct	yearlyWine	
specific	producerWithGeoLocation		producerWithImage		
	productAttributeWithImage		wineProductWithGeoLocation		
relation	hasAttribute				

With these entities and relations we created the ORM2 diagram of Figure 2.33. We added additional value constraints (Subsection 2.1.2.1) based on the data available.

appellation

t_i	**AID**	AN	RID
t_1	4	Provence	2
t_2	1	Veneto	1
t_3	2	Piedmont	1

region

t_i	**RID**	RN
t_1	2	France
t_2	1	Italy
t_3	1	Italy

varietalBlend

t_i	**VBID**	VBN	TID
t_1	100	Rosé	3
t_2	3	Rosé	1
t_3	4	Nebbiolo	2

producer

t_i	**PID**	PN	PU	PIU	PG
t_1	1	La Bastide Blanche	.../w=1	w1.jpg	.../wnmap=1
t_2	3	Lamberti	.../w=3	w3.jpg	
t_3	2	Cascina Adelaide	.../w=2		

wine

t_i	**WN**	PID
t_1	La BBB Rosé	1
t_2	Lamberti Rose Spumante	3
t_3	C. Adelaide	2

wineType

t_i	**TID**	TN
t_1	3	Rosé Wine
t_2	1	Sparkling
t_3	2	Red Wines

yearlyWine

t_i	**WN**	**WPY**	AID	VBID
t_1	La BBB Rosé	2004	4	100
t_2	Lamberti Rose Spumante	NV	1	3
t_3	C. Adelaide	2010	2	4

wineProduct

t_i	**WPID**	WN	WPY	BT	WPG
t_1	1	La BBB Rosé	2004	750ml	.../map=1
t_2	2	Lamberti Rose Spumante	NV	750ml	
t_3	3	C. Adelaide	2010	1.5L	

wineProduct

t_i	WPU	WPM	WPX	WPR	WPT	WPV
t_1	.../detail=1	2.8	6.7	2.8	90	5
t_2	.../detail=2	3.3	7.3	5.8	0	0
t_3	.../detail=3	9.6	9.6	9.6	97	0

productAttribute

t_i	**PAID**	PAIU	PAN
t_1			
t_2			
t_3	2		Bold

hasAttribute

t_i	**WPID**	**PAID**
t_1	1	
t_2	2	
t_3	3	2

Figure 2.29: Sample in 4NF

appellation

t_i	**AID**	AN	RID
t_1	4	Provence	2
t_2	1	Veneto	1
t_3	2	Piedmont	1

region

t_i	**RID**	RN
t_1	2	France
t_2	1	Italy
t_3	1	Italy

varietalBlend

t_i	**VBID**	VBN	TID
t_1	100	Rosé	3
t_2	3	Rosé	1
t_3	4	Nebbiolo	2

producer

t_i	**PID**	PN	PU
t_1	1	La Bastide Blanche	.../w=1
t_2	3	Lamberti	.../w=3
t_3	2	Cascina Adelaide	.../w=2

wine

t_i	**WN**	PID
t_1	La BBB Rosé	1
t_2	Lamberti Rose Spumante	3
t_3	C. Adelaide	2

wineType

t_i	**TID**	TN
t_1	3	Rosé Wine
t_2	1	Sparkling
t_3	2	Red Wines

yearlyWine

t_i	**WN**	**WPY**	AID	VBID
t_1	La BBB Rosé	2004	4	100
t_2	Lamberti Rose Spumante	NV	1	3
t_3	C. Adelaide	2010	2	4

wineProduct

t_i	**WPID**	WN	WPY	BT
t_1	1	La BBB Rosé	2004	750ml
t_2	2	Lamberti Rose Spumante	NV	750ml
t_3	3	C. Adelaide	2010	1.5L

wineProduct

t_i	WPU	WPM	WPX	WPR	WPT	WPV
t_1	.../detail=1	2.8	6.7	2.8	90	5
t_2	.../detail=2	3.3	7.3	5.8	0	0
t_3	.../detail=3	9.6	9.6	9.6	97	0

productAttribute

t_i	**PAID**	PAN
t_3	2	Bold

hasAttribute

t_i	**WPID**	**PAID**
t_3	3	2

wineProductWithGeoLocation

t_i	**WPID**	WPG
t_1	1	.../map=1

producerWithGeoLocation

t_i	**PID**	PG
t_1	1	.../wnmap=1

producerWithImage

t_i	**PID**	PIU
t_1	1	w1.jpg
t_2	3	w3.jpg

Figure 2.30: Sample after null processing

$winePoduct(\underline{WPID}, \underline{\underline{YWID, BT}}, WPU, WPM, WPX, WPR, WPT, WPV)$
$winePoductWithGeoLocation(\underline{WPID}, WPG)$
$productAttributeWithImage(\underline{PAID}, PAIU)$
$yearlyWine(\underline{YWID}, \underline{\underline{WID, WPY}}, AID, VBID)$

$appellation(\underline{AID}, AN, RID)$
$hasAttribute(\underline{WPID, PAID})$
$producer(\underline{PID}, PN, PU)$
$producerWithImage(\underline{PID}, PIU)$
$producerWithGeoLocation(\underline{PID}, PG)$
$productAttribute(\underline{PAID}, PAN)$
$region(\underline{RID}, RN)$
$varietalBlend(\underline{VBID}, VBN, TID)$
$wine(\underline{\underline{WID, WN}}, PID)$
$wineType(\underline{TID}, TN)$

Figure 2.31: Final schema

appellation

t_i	**AID**	AN	RID
t_1	4	Provence	2
t_2	1	Veneto	1
t_3	2	Piedmont	1

region

t_i	**RID**	RN
t_1	2	France
t_2	1	Italy
t_3	1	Italy

varietalBlend

t_i	**VBID**	VBN	TID
t_1	100	Rosé	3
t_2	3	Rosé	1
t_3	4	Nebbiolo	2

producer

t_i	**PID**	PN	PU
t_1	1	La Bastide Blanche	.../w=1
t_2	3	Lamberti	.../w=3
t_3	2	Cascina Adelaide	.../w=2

wine

t_i	***WID***	WN	PID
t_1	*1*	La BBB Rosé	1
t_2	*2*	Lamberti Rose Spumante	3
t_3	*3*	C. Adelaide	2

wineType

t_i	**TID**	TN
t_1	3	Rosé Wine
t_2	1	Sparkling
t_3	2	Red Wines

yearlyWine

t_i	***YWID***	*WID*	WPY	AID	VBID
t_1	*y1*	*1*	2004	4	100
t_2	*y2*	*2*	NV	1	3
t_3	*y3*	*3*	2010	2	4

wineProduct

t_i	**WPID**	*YWID*	BT
t_1	1	*y1*	750ml
t_2	2	*y2*	750ml
t_3	3	*y3*	1.5L

wineProduct

t_i	WPU	WPM	WPX	WPR	WPT	WPV
t_1	.../detail=1	2.8	6.7	2.8	90	5
t_2	.../detail=2	3.3	7.3	5.8	0	0
t_3	.../detail=3	9.6	9.6	9.6	97	0

productAttribute

t_i	**PAID**	PAN
t_3	2	Bold

hasAttribute

t_i	**WPID**	**PAID**
t_3	3	2

wineProductWithGeoLocation

t_i	**WPID**	WPG
t_1	1	.../map=1

producerWithGeoLocation

t_i	**PID**	PG
t_1	1	.../wnmap=1

producerWithImage

t_i	**PID**	PIU
t_1	1	w1.jpg
t_2	3	w3.jpg

Figure 2.32: Final schema for the sample tuples

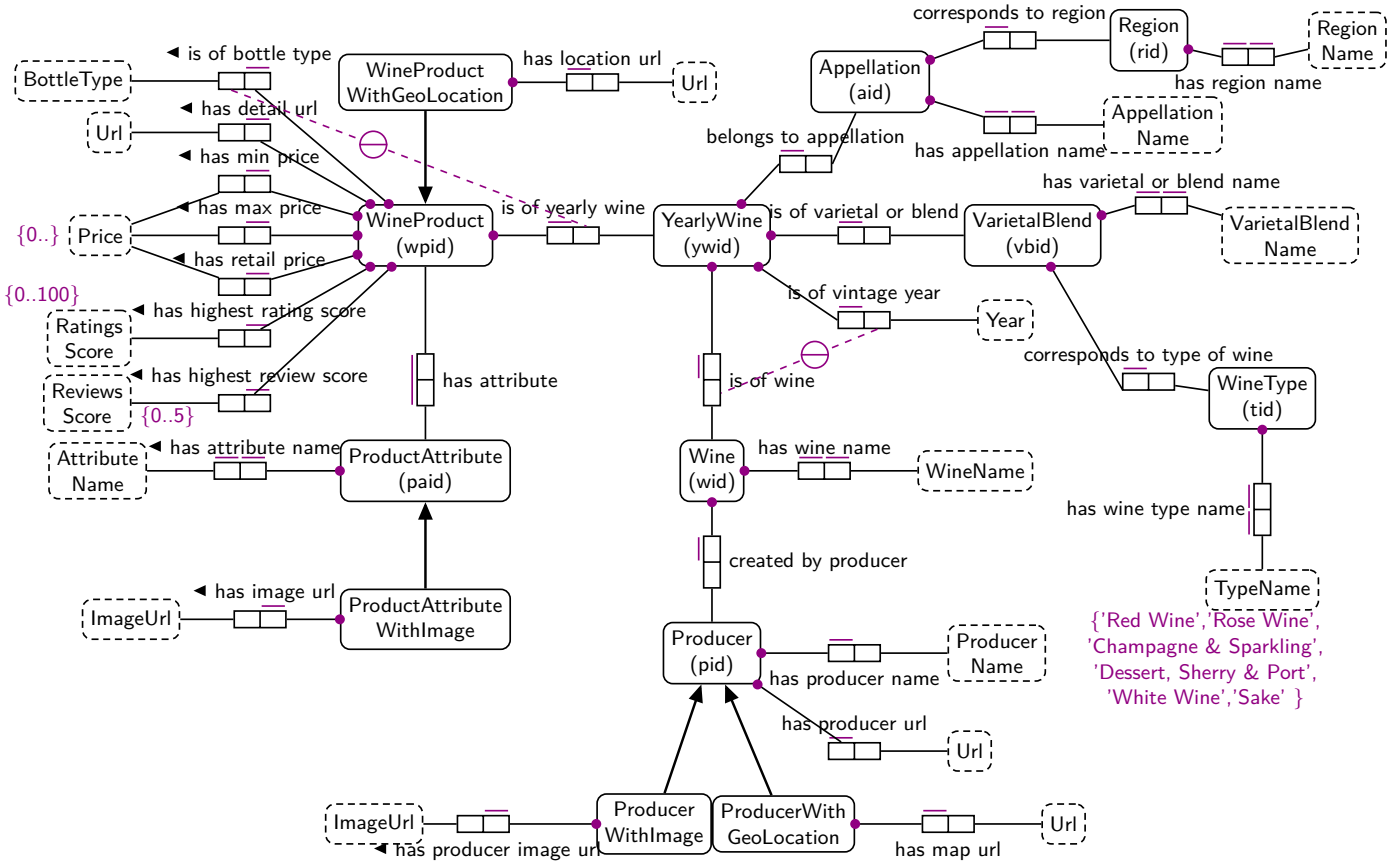


Figure 2.33: Final ORM2 diagram

Chapter 3

From ORM2 to OWL 2

One the results of this thesis is an OWL ontology with data. This ontology should preserve the semantics of the ORM2 conceptual model obtained from the raw data. Therefore, we developed a semantics preserving mapping between ORM2 and OWL 2.

To formalize this mapping we use as middle man description logics (DL). How are these two knowledge representation tools related to DL? The semantics of OWL 2 are compatible with the description logic $\mathcal{SROIQ}^{(\mathcal{D})}$ [29]. And for ORM2 there exist attempts to formalize it into a DL [8, 20, 22, 24].

To automatize the mapping process, first we should adopt a textual syntax for ORM2. Some syntaxes have been proposed [6, 8, 31, 33], but there is not yet a standard syntax to our knowledge. Demey et al. [6] proposed a markup syntax in XML. Pan and Liu [31] presented an update of this syntax, extending it to support the new features of ORM2 and making it less redundant.

Pan and Liu also proposed an abstract syntax based in their metamodel for ORM2 [32]. Franconi and Mosca [8] proposed a linear syntax with set-theoretic semantics called ORM2$^{\text{plus}}$ which covers most of the introduced constructs and constraints of an ORM2 model. We will adopt this syntax and use the defined semantics to create a formal mapping from ORM2 to OWL 2.

One important requirement of the resulting ontology is to be compatible?? with Quelo. In Quelo queries can be posed to an ontology by forming a natural language query via menus. Thus, our ontology should be verbalizable by the Natural Language Generation engine of Quelo . That is, the names assigned to the elements of the ontology should be descriptive.

Before describing ORM2$^{\text{plus}}$ syntax, we will briefly compare ORM2 with OWL 2 regarding the assumptions they make about the world, names and hierarchy (Section 3.1). Afterwards we will ??describe?? Description Logics in Section 3.2. Then we will then present the ORM2$^{\text{plus}}$ syntax, the core fragment ORM2$^{\text{zero}}$ and the fragment ORM2$^{\text{bin}}$ in Section 3.3. In Section 3.4 we will describe the DL semantics of ORM2$^{\text{bin}}$. The functional syntax and semantics of OWL 2 are introduced?? in Section 3.5. In the last section, Section 3.6, we will provide the mapping from the fragment ORM2$^{\text{bin}}$ to OWL 2.

3.1 ORM2 vs OWL 2 assumptions

3.1.1 World assumptions

Regarding the completeness of the data, ORM2 being a modelling tool for databases, makes the *Closed World Assumption* (CWA) [27], i.e., anything that cannot be proven true is assumed to be false. The data is considered to be complete; therefore, only the available data is true. OWL on the other hand, as the language for the Semantic Web, was designed to deal with incomplete information. Instead of the CWA, OWL uses the *Open World Assumption* (OWA) [28] where anything that cannot be proven true is not false but unknown. In other words, the failure to prove something true does not imply it is false.

3.1.2 Name assumptions

Another difference between ORM2 and OWL 2 is their assumption regarding names. ORM2 also makes the *Unique Name Assumption* (UNA) where different names refer to different individuals. OWL 2 does not make the UNA [28] and different names can refer to the same individual. UNA in OWL 2 can be axiomatized by explicitly declaring which individuals are different from which others.

3.1.3 Hierarchy relation

ORM2 differs from OWL also in the implementation of hierarchy between object types/classes, subtyping in ORM2 and subclassing in OWL 2. OWL 2 subclassing is based in the semantics of RDFS which is based in the semantics of the subset relation [19]. Therefore, subclassing in OWL 2 is both *reflexive* and *transitive*, which means that any class is a subclass of itself and for any classes A,B, and C such that A is subclass of class B and B is subclass of C it holds that A is subclass of C. However, in ORM2 subtyping is irreflexive and intransitive [14].

3.1.4 Some remarks about names

Up till now we have mainly discussed about the ORM2 elements; however, from this section on, we will include the elements of DL and OWL 2 and naming clashes will ensue.

ORM2 divides the world into objects types, which can be entity or value types; and roles, which are the basic unit to form predicates [12]. Object types and roles are populated by instances. OWL 2 elements include *entities*: classes, datatypes, properties, and named individuals; anonymous individuals; and literals [28]. Properties can be object properties, data properties, or annotation properties. Finally, the elements of DL are concepts, *roles*, and individuals [1, chap. 2]. Table 3.1 shows an example of how the elements of the sentence "Jane is married to a person born in the year 1810." are modelled with these three formalisms.

3.2 Description Logics

Description Logics (DL) are a family of knowledge representation languages which have logic based semantics [1, Chap. 2]. This formal semantics allows

Example	ORM2	OWL 2	DL
Jane	instance	individual	individual
is married to	predicate	object prop.	role
a person	entity type	class	concept
born in	predicate	data prop.	role
the year	value type	datatype	concept
1810	instance	literal	individual

Table 3.1: Model in ORM2, OWL 2, and DL.

one to reason over the described knowledge. Furthermore, DLs are a *decidable* fragment of First-Order Logic.

The basic elements of any DL are *concepts*, *roles*, and *individuals*; similar to object types, predicates, and populations in ORM. The semantics of DL is set-theoretic, thus each concept is interpreted as a set of individuals and each role as a set of pairs of individuals. For example, to model the domain of book writers the set of all authors can be represented by the concept Author and the set of all books by the concept Book. The role wrote can connect Authors with the Books they wrote. The individual melville which represents the author Herman Melville will be in the class Author. And the tuple (melville, moby-dick) should be part of the extension of the role wrote.

The representation of the world using DL is partial. DLs represent the world with *axioms*, which are statements that are true in the description, but all the rest is unknown, in agreement with the OWA.

These axioms are divided into three groups: terminological axioms, assertional axioms and relational axioms. The terminological axioms, also called TBox axioms, describe relations between concepts. The assertional axioms, or ABox axioms, describe the relations between individuals and the concepts to which they belong. Finally the relational axioms, RBox axioms, state the relations between roles. Together these boxes for the knowledge base of a DL ontology.

Before introducing the constructors for the axioms, we introduce the constructors for concepts and roles.

3.2.1 Concept and role constructors

Within the attributive language ($\mathcal{AL}$) DL one can create concepts using these constructors:

$$C, D \to A \mid \top \mid \bot \mid \neg A \mid C \sqcap D \mid \forall R.C \mid \exists R.\top,$$

where A is an atomic concept, R an atomic role and, C and D are *concept descriptions*. So the possible concept descriptions for $\mathcal{AL}$ are atomic concepts; $\top$, the universal concept; $\bot$, the bottom concept; atomic negation; intersection of concepts; value restriction; and limited existential quantification.

The semantics of any DL is based in the notion of an interpretation $\mathcal{I}$, which is formed by a non empty set $\Delta^{\mathcal{I}}$ called the domain of the interpretation and an interpretation function $\cdot^{\mathcal{I}}$. This function assigns to each atomic concept A a set of elements of the domain and to each atomic role R a binary relation in

Name	Syntax	Semantics
$\mathcal{U}$, union of concepts	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\mathcal{C}$, full negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$\mathcal{E}$, full existential quantification	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
$\mathcal{N}$, number restrictions	$\geq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}}\} \geq n\}$
	$\leq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}}\} \leq n\}$
$\mathcal{Q}$, qualified number restrictions	$\geq nR.C$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq n\}$
	$\leq nR.C$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq n\}$
Local reflexivity	$\exists R.Self$	$\{x \mid (x,x) \in R^{\mathcal{I}}$
$\mathcal{O}$, nominals	$\{a\}$	$\{a^{\mathcal{I}}\}$
$\mathcal{I}$, inverse role	R^{-}	$\{(y,x) \mid (x,y) \in R^{\mathcal{I}}\}$
role intersection	$R \sqcap S$	$R^{\mathcal{I}} \cap S^{\mathcal{I}}$

Table 3.2: DL concept and role constructors

$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Thus, the semantics of each of the $\mathcal{AL}$ constructors is the following:

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg A)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} \\
(C \sqcap D) &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \forall y.(x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\} \\
(\exists R.\top)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \exists y.(x,y) \in R^{\mathcal{I}}\}
\end{aligned}$$

The set of concept and role constructors characterize a DL. In Table 3.2 we show the name, syntax, and semantics for some DL constructors. The symbol # is used to denote the cardinality of a set. The name of each DL is roughly based on the letters of its constructors. For example, the $\mathcal{ALCQI}$ DL includes, besides the basic constructors of $\mathcal{AL}$, the constructors for full negation ($\mathcal{C}$), qualified number restrictions ($\mathcal{Q}$), and inverse roles ($\mathcal{I}$).

3.2.2 Axioms

The axioms of the TBox are called *general concept inclusion* axioms (GCI) and are of the form $C \sqsubseteq D$, where C and D are complex concepts. If $C \sqsubseteq D$, and $D \sqsubseteq C$ hold in a TBox these axioms can be combined into a *concept equivalence* axiom $C \equiv D$. An interpretation $\mathcal{I}$ satisfies an inclusion $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and satisfies an equivalence $C \equiv D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$. For a TBox $\mathcal{T}$, $\mathcal{I}$ satisfies $\mathcal{T}$ iff $\mathcal{I}$ satisfies every axiom of $\mathcal{T}$.

Assertional axioms are of two forms *concept assertions*, $C(a)$, and *role assertions*, $R(b,c)$, where C is a concept, R a role and a, b, and c are individuals. An interpretation $\mathcal{I}$ satisfies $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and satisfies $R(b,c)$ if $(b^{\mathcal{I}}, c^{\mathcal{I}}) \in R^{\mathcal{I}}$. If every axiom of ABox $\mathcal{A}$ is satisfied by an interpretation $\mathcal{I}$, then this interpretation satisfies the ABox $\mathcal{A}$.

Relational axioms include *role inclusion*, $R \sqsubseteq S$ and *role equivalence*, $R \equiv S$, where R and S are atomic roles [25]. Role inclusions create *role hierarchies*, and ontologies with role hierarchies add the letter $\mathcal{H}$ to their DL name. An

interpretation $\mathcal{I}$ satisfies $R \sqsubseteq S$ if $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$, and satisfies $R \equiv S$ if $R^{\mathcal{I}} = R^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies a RBox $\mathcal{R}$ iff it satisfies all the axioms in $\mathcal{R}$.

In the following section we will use these notions to stablish the formal semantics for ORM2$^{\text{plus}}$. Also the semantics of OWL 2 are based on the $\mathcal{SROIQ}^{(\mathcal{D})}$ DL and we use the notions of this section to introduce?? the OWL 2 syntax and semantics.

3.3 ORM2$^{\text{plus}}$ syntax

While an ORM diagram is useful to visualize the conceptual schema, it is also important to detect inconsistencies in the design. The verbalisation of the diagram offers an insight of its meaning; however, to reason with the constraints in the diagram, a formal specification of the ORM2 *semantics* is necessary. But, before specifying the semantics, one should specify the *syntax*.

The linear syntax ORM2$^{\text{plus}}$ gives for each ORM2 construct one or more predicates which have the same meaning [8]. The signature $\mathcal{S}$ of ORM2$^{\text{plus}}$ is a tuple $(\mathcal{E}, \mathcal{V}, \mathcal{R}, \mathcal{A}, \mathcal{D}, \Lambda, \Lambda_{(\cdot)}, \varrho, \tau)$, where

- $\mathcal{E}$ is a set of entity type symbols,
- $\mathcal{V}$ is a set of value type symbols,
- $\mathcal{R}$ is a set of relation symbols[1],
- $\mathcal{A}$ is a set of role symbols,
- $\mathcal{D}$ is a set of domain symbols,
- Λ is a set of pairwise disjoint sets of values,
- for each $D \in \mathcal{D}$, $\Lambda_{(\cdot)} : \mathcal{D} \to \Lambda$ is an injective extension function associating each domain symbol D to an extension Λ_D
- $\varrho \subseteq \mathcal{R} \times \mathcal{A}$ is a binary relation linking role symbols to relation symbols. Given a relation symbol R and a role symbol a the pair $R.a$ is called a *localised role*. For each $R \in \mathcal{R}$, $\varrho_R = \{R.a \mid R.a \in \varrho\}$
- for each $R \in \mathcal{R}$, $\tau_R : \varrho_R \to [1..|\varrho_R|]$ is a bijection mapping localised roles to argument positions in a relation. We define $\tau = \bigcup_{R \in \mathcal{R}} \tau_R$

An ORM2$^{\text{plus}}$ conceptual schema ξ over a signature $\mathcal{S}$ is a tuple $\langle \mathcal{E}, \mathcal{V}, \mathcal{R}, \mathcal{C} \rangle$ where $\mathcal{E}$,$\mathcal{V}$, and $\mathcal{R}$ are the entity types, value types and relations in ξ and $\mathcal{C}$ is a finite set of the following constructs. $\wp(A)$ denotes the power set of A.

1. $\mathsf{TYPE} \subseteq \varrho \times (\mathcal{E} \cup \mathcal{V})$
2. $\mathsf{FREQ} \subseteq \wp(\varrho) \times (\wp(\varrho) \times \wp(\varrho)) \times (\mathbb{N} \times (\mathbb{N} \cup \{\infty\}))$
3. $\mathsf{MAND} \subseteq \wp(\varrho) \times (\mathcal{E} \cup \mathcal{V})$
4. $\mathsf{R\text{-}SET_H} \subseteq \big(\wp(\varrho) \times (\wp(\varrho) \times \wp(\varrho))\big) \times \big(\wp(\varrho) \times (\wp(\varrho) \times \wp(\varrho))\big) \times (\mu : \varrho \to \varrho)$, $\mathsf{H} = \{\mathsf{Sub}, \mathsf{Exc}\}$
5. $\mathsf{O\text{-}SET_H} \subseteq \wp(\mathcal{E} \cup \mathcal{V}) \times (\mathcal{E} \cup \mathcal{V})$, $\mathsf{H} = \{\mathsf{Isa}, \mathsf{Tot}, \mathsf{Ex}\}$
6. $\mathsf{O\text{-}CARD} \subseteq (\mathcal{E} \cup \mathcal{V}) \times (\mathbb{N} \times (\mathbb{N} \cup \{\infty\}))$

[1] In this section we use the term *relation* to denote ORM predicates.

7. R-CARD $\subseteq \varrho \times (\mathbb{N} \times (\mathbb{N} \cup \{\infty\}))$
8. OBJ $\subseteq \mathcal{R} \times (\mathcal{E} \cup \mathcal{V})$
9. RING$_{\mathsf{J}} \subseteq \varrho \times \varrho$, $\mathsf{J} = \{\mathsf{Irr}, \mathsf{Asym}, \mathsf{Trans}, \mathsf{Intr}, \dots\}$
10. V-VAL $\subseteq \mathcal{V} \to \wp(\Lambda_D)$ for some $\Lambda_D \in \Lambda$

These ten constructs cover most of the ORM2 constraints introduced in Section 2.1. TYPE binds the roles to its object type forming fact types. FREQ covers both external and internal frequency constraints, and MAND the external and internal mandatory constraints. The subset and set exclusion constraints are represented with R-SET$_{\mathsf{H}}$. Subtyping, total subtypes, and exclusive subtypes are indicated with O-SET$_{\mathsf{H}}$. O-CARD and R-CARD are for object cardinality constraints and role cardinality constraints, respectively. The OBJ construct specifies objectification and RING$_{\mathsf{J}}$ is for all the ring constraints. Finally, V-VAL specifies object value constraints.

Six ORM2 constraints are considered derived constraints, i.e., can be formulated with combinations of the ten basic constructs. Uniqueness constraints, internal and external, are obtained using the FREQ construct restricting the maximum and minimum to one. Set equality constraints are two subset constraints which have the last parameter μ inverted. The exclusive-or constraint can be obtained with the constructs R-SET$_{\mathsf{Exc}}$ and MAND. The subtypes partition constraint is the sum of exclusive and total subtypes; and we use O-SET$_{\mathsf{Tot}}$ and O-SET$_{\mathsf{Ex}}$ to represent it. Lastly, role value constraints can be achieved by introducing a new value type symbol V^* and using TYPE to link it to the constrained role and V-VAL to constrain the role values.

The only constraint not considered in ORM2$^{\text{plus}}$ is the value comparison constraint (Subsec. 2.1.4). The preferred unique constraint is not considered in ORM2$^{\text{plus}}$ as it is contained in the semantics of unique (frequency) and mandatory constraints.

In the linear syntax the sets $\mathcal{E}$,$\mathcal{V}$, and $\mathcal{R}$ are explicitly listed in the constructs ENTITYTYPES, VALUETYPES, and RELATIONS, respectively. For completeness we also provide the extended Backus-Naur Form grammar for ORM2 in Appendix C.

In the following subsection we will give examples for each of the constraints. We omit the types and relations declaration for space reasons. We use some of the examples introduced in Section 2.1, and we repeat here the figures for ease of reference.

3.3.1 Value constraints

In Listing 3.1 we show the linear syntax for the ORM2 diagram of Figure 2.6. The first two lines declare the association between the relation and the object types of the fact type. Line 3 states the uniqueness constraint over the first role of the relation. The object value constraint over *Month* is defined in line 4. To represent the role value constraint first associate a new value type with the role (line 6) and then we restrict this new value type (line 7).

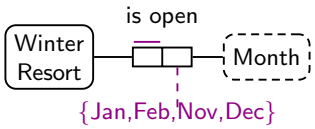


Figure 2.6: An ORM2 diagram with a role value constraint.

```
1 TYPE(isOpen.winterResort, WinterResort)
2 TYPE(isOpen.month, Month)
3 FREQ({isOpen.winterResort},{},<1,1>)
4 V-VAL(Month) = {'Jan','Feb','Mar','Apr',...}
5 % WinterMonth is a new value type symbol
6 TYPE(isOpen.month, WinterMonth)
7 V-VAL(WinterMonth) = {'Jan','Feb','Nov','Dec'}
```

Listing 3.1: ORM2^{plus} value constraints.

3.3.2 Cardinality constraints

The linear syntax construct for the example in Figure 2.5 is shown in Listing 3.2 line 1. On the RHS of the construct are the upper and lower bounds for the cardinality of the object. Lines 2 and 3 of the same listing are the linear representation of the diagram on Figure 2.7. Line 2 states the object type associated with the role and line 3 the cardinality constraint.

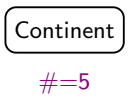


Figure 2.5: An ORM2 object type with a cardinality constraint.

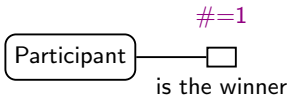


Figure 2.7: An ORM2 diagram with a role cardinality constraint.

```
1 O-CARD(Continent)=(5,5)
2 TYPE(isTheWinner.participant,Participant)
3 R-CARD(isTheWinner.participant)=(1,1)
```

Listing 3.2: ORM2^{plus} cardinality constraints.

3.3.3 Mandatory and set-comparison constraints

For Figure 2.18, Listing 3.3 has the translation of the diagram into ORM2^{plus}. The first eight lines state the role associations and their frequency in the relations. On line 10 we state the mandatory constraint over the first roles of both relations and line 11 the exclusion set constraint for the same roles. Together they construct the exclusive-or constraint.

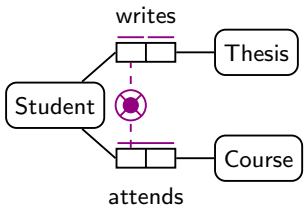


Figure 2.18: An ORM2 diagram with an exclusive-or constraint.

```
1  TYPE(writes.thesis, Thesis)
2  TYPE(writes.student, Student)
3  TYPE(attends.course, Course)
4  TYPE(attends.student, Student)
5
6  FREQ({writes.thesis},{},<1,1>)
7  FREQ({writes.student},{},<1,1>)
8  FREQ({attends.course, attends.student},{},<1,1>)
9  % exclusive-or constraint
10 MAND({writes.student,attends.student},Student)
11 R–SET_Exc(({writes.student}, {}),({attends.student},
       {}),{(writes.student,attends.student)})
```

Listing 3.3: ORM2^{plus} exclusive-or constraint.

3.3.4 Ring constraints

Figure 2.22c shows a fact type with an acyclic intransitive ring constraint which in ORM2^{plus} is described as shown in Listing 3.4.

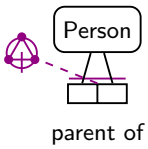


Figure 2.22c: An ORM2 diagram with an acyclic intransitive ring constraint.

```
1  TYPE(parentOf.sub,Person)
2  TYPE(parentOf.obj,Person)
3  FREQ({parentOf.sub,parentOf.obj},{},<1,1>)
4  % Acyclic intransitive ring constraint
5  RING_Acyclic(parentOf.sub, parentOf.obj)
6  RING_Intr(parentOf.sub, parentOf.obj)
```

Listing 3.4: ORM2^{plus} ring constraint.

3.3.5 Subtyping and subtype constraints

Simple subtyping, as shown in Figure 2.23a is represented in ORM2^{plus} with the O-SET$_{Isa}$ construct (Listing 3.5 line 1). To construct a subtype partition constraint as in Fig. 2.23d, we derive it from the constructs O-SET$_{Tot}$ and O-SET$_{Ex}$ (Listing 3.5 lines 3, 4).

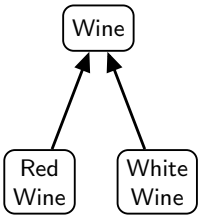


Figure 2.23a: An ORM2 diagram with subtyping.

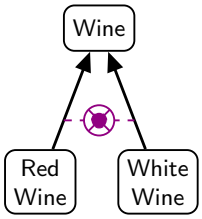


Figure 2.23d: An ORM2 diagram with subtype partition constraint.

```
1 O–SET_Isa({RedWine, WhiteWine}, Wine)
2 % Subtype partition
3 O–SET_Tot({RedWine, WhiteWine}, Wine)
4 O–SET_Ex({RedWine, WhiteWine}, Wine)
```

Listing 3.5: ORM2^{plus} subtyping.

3.3.6 Objectification

Listing 3.6 is the representation of the diagram in Figure 2.25. The construct OBJ represents objectification in ORM2^{plus}, as shown in line 5.

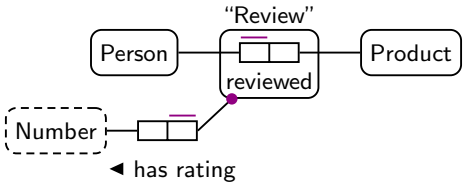


Figure 2.25: An ORM2 diagram with objectification.

```
1 TYPE(reviewed.person, Person)
2 TYPE(reviewed.product, Product)
3 FREQ({reviewed.person},{},<1,1>)
4 % Objectification
5 OBJ(reviewed, Review)
6 TYPE(hasRating.review, Review)
7 TYPE(hasRating.number, Number)
8 FREQ({hasRating.review},{},<1,1>)
9 MAND({hasRating.review}, Review)
```

Listing 3.6: ORM2^{plus} objectification.

3.3.7 Join constraints

Up till now all of the examples contained simple external constraints, but for the subset constraint in Figure 2.24, we need a join constraint. The translation of this diagram to ORM2^{plus} is Listing 3.7. The declaration of the subset constraint in is line 15. Let is recall the formal definition of the R-SET$_H$ construct.

$$\text{R-SET}_H \subseteq \underbrace{\big(\overbrace{\wp(\varrho)}^{R_A} \times \overbrace{(\wp(\varrho) \times \wp(\varrho))}^{J_A}\big)}_{A} \times \underbrace{\big(\overbrace{\wp(\varrho)}^{R_B} \times \overbrace{(\wp(\varrho) \times \wp(\varrho))}^{J_B}\big)}_{B} \times \underbrace{(\mu : \varrho \to \varrho)}_{\mu}$$

Informally, this construct represents the constraint $A \subseteq B$. R_A and R_B contain the list of roles involved in the constraint. J_A and J_B define how the roles in each R_i are related. In this example, $R_A = \{isCreatedIn.country, isFromRegion.region\}$. These two localised roles are related via *Wine*, so J_A contains the tuple $\langle isCreatedIn.wine = isFromRegion.wine \rangle$. J_B in the other hand, is empty as the roles in R_B are in the same relation and there is no need of joins. Finally, μ states how these two lists of localised roles are related to each other, for this constraint which role is subset of which role. For our example μ has two tuples $(isCreatedIn.country, locatedIn.country)$ and $(isFromRegion.region, locatedIn.region)$. All these elements define a subset join constraint.

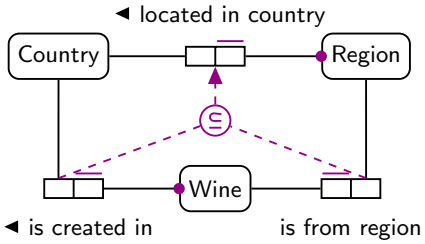


Figure 2.24: An ORM2 diagram with a join subset constraint.

```
1  TYPE(locatedIn.region,Region)
2  TYPE(locatedIn.country,Country)
3  FREQ({locatedIn.region},{},<1,1>)
4  MAND({locatedIn.region}, Region)
5
6  TYPE(isCreatedIn.wine,Wine)
7  TYPE(isCreatedIn.country,Country)
8  FREQ({isCreatedIn.wine},{},<1,1>)
9  MAND({isCreatedIn.wine}, Wine)
10
11 TYPE(isFromRegion.wine,Wine)
12 TYPE(isFromRegion.region,Region)
13 FREQ({isFromRegion.wine},{},<1,1>)
14 % Join subset constraint
15 R-SET_Sub(({isCreatedIn.country,isFromRegion.region
       },{<isCreatedIn.wine=isFromRegion.wine>}),
       ({locatedIn.country,locatedIn.region},{}),
       {(isCreatedIn.country,locatedIn.country),
       (isFromRegion.region,locatedIn.region)})
```

Listing 3.7: ORM2^{plus} join constraint.

3.3.8 ORM2^{zero} syntax

Besides ORM2^{plus}, Franconi and Mosca [8] identified a core fragment of ORM2 which can be encoded into the ExpTime-complete description logic (DL) $\mathcal{ALCQI}$. This fragment has these constructs:

1. TYPE
2. FREQ$^{-} \subseteq \varrho \times (\mathbb{N} \times (\mathbb{N} \cup \{\infty\}))$
3. MAND

4. R-SET$^{-}_{\mathsf{H}} \subseteq \wp(\varrho) \times \wp(\varrho)$, $\mathsf{H} = \{\mathsf{Sub}, \mathsf{Exc}\}$
5. O-SET$_{\mathsf{H}}$, $\mathsf{H} = \{\mathsf{Isa}, \mathsf{Tot}, \mathsf{Ex}\}$
6. OBJ

TYPE, MAND, O-SET$_{\mathsf{H}}$, and OBJ are defined as in ORM2$^{\mathsf{plus}}$. FREQ^{-} is a restriction of FREQ which can be applied only to single roles. R-SET$^{-}_{\mathsf{H}}$ accepts only two single roles or two whole relations of the same arity.

The semantics introduced by Franconi and Mosca [8] for ORM2$^{\mathsf{zero}}$ use *reification* to overcome the lack of n-ary relations in $\mathcal{ALCQI}$, i.e., for each relation R of arity n a new atomic entity A_R and n functional roles $\tau(R.a_i)$ are introduced. An example of reification is shown in Figure 3.1.

(a) Ternary relation.

(b) Reified relation.

Figure 3.1: Reification.

To obtain clearer?? verbalizations in Quelo we want?? to avoid reification. Reifing the relation makes the verbalizations repetitive and unnatural because the generated names for the new entity and the functional roles are artificial names not found originally in the conceptual model. Sometimes these names are part of the UoD, but not always. These artificial names will reduce the fluency of the verbalizations in Quelo. For example, a possible verbalization for the reified relation in Figure 3.1b could be "I am looking for a grade course which should be for a course.". (What is a "grade course"?) Therefore, we propose a new ORM2 fragment obtained?? by restricting ORM2$^{\mathsf{zero}}$ to binary relations. We call this fragment ORM2$^{\mathsf{bin}}$.

3.3.9 ORM2$^{\mathsf{bin}}$ syntax

The allowed constructs for ORM2$^{\mathsf{bin}}$ are the following:

1. TYPE
2. FREQ^{-}
3. MAND
4. R-SET$^{-}_{\mathsf{H}}$, $\mathsf{H} = \{\mathsf{Sub}, \mathsf{Exc}\}$
5. O-SET$_{\mathsf{H}}$, $\mathsf{H} = \{\mathsf{Isa}, \mathsf{Tot}, \mathsf{Ex}\}$

We disallow the OBJ construct as well because it generates the same naming problems as reification. The semantics for this ORM2 fragment will be described in the next section. We include the translation of our schema into ORM2$^{\mathsf{bin}}$ syntax in Appendix D.

3.3.9.1 Preferred UC

We should note that ORM2plus considers the reference mode constraints as simple uniqueness constraints. While the semantics of these constraints are covered by the FREQ⁻ and MAND constructs, that the designer of the conceptual schema chose this specific relation as the preferred identifier should be represented in the syntax as well. There may be more than one unique mandatory relations for an entity, but just one is the reference mode. To indicate this choice we add the construct PRIMARY $\subseteq \mathcal{E} \times \varrho$ to denote which role is the preferred identifier. This construct is just a flag, and does not affect the semantics.

3.3.10 Names and labels

Most of the entities in the obtained ORM2 diagram have, besides the preferred reference mode, a mandatory relation to its name, as in Figure 3.2. We add a second construct LABEL $\subseteq \mathcal{E} \times \varrho$ to indicate which is the role which has the name of the entity. We use this value as the rdfs:label annotation property for each individual, which according to the OWL2 Structural Specification "can be used to provide an IRI with a human-readable label" [28]. This decision provided us with two benefits. First, our ontology is not polluted with repetitive relations of the type hasXName, and therefore is more concise. And second, this label will produce better verbalisations. It is important to note that annotations are not included in the reasoning process [28]. However, the existence of these relation is meaningful for the exploration of the ontology.

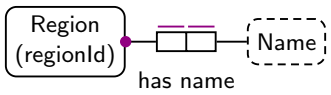


Figure 3.2: Entity with hasName

3.4 ORM2bin semantics

To create a semantics preserving mapping from ORM2 to OWL 2 we need the logic representation of the ORM2 schema.

In 1989 Halpin [11] gave the first full formalization for ORM using FOL-based semantics. Since then the ORM has evolved and increased, but not so the formalization. Others have proposed formalizations of the ORM diagrams into diverse logics [8, 21, 22, 24]. Jarrar [21] and Keet [24] proposed a formalization into the $\mathcal{DLR}_{ifd}$ description logic, which is a DL with n-ary relations. Jarrar [22] also proposed a translation of ORM into $\mathcal{SHOIN}$, the DL underlying OWL DL.

Franconi and Mosca [8] provided for each of the constructs of ORM2plus set-theoretic semantics based in relational algebra and also FOL semantics. For the fragment ORM2zero they provided semantics in $\mathcal{ALCQI}$ and $\mathcal{DLR}_{ifd}$. Based on these semantics for ORM2plus and ORM2zero [8, 10], we obtain the DL semantics for ORM2bin.

The DL semantics for ORM2bin use of the following S^{DL} signature:

Syntax	Semantics
$\mathsf{TYPE}(R.a, O)$	$\exists \hat{R}_a \sqsubseteq O$
$\mathsf{FREQ}^{-}(R.a, \langle min, max \rangle)$	$\exists \hat{R}_a \sqsubseteq \geq min\, \hat{R}_a \sqcap \leq max\, \hat{R}_a$
$\mathsf{MAND}(\{R^1.a_1, \ldots, R^k.a_k\}, O)$	$O \sqsubseteq \exists \hat{R^1}_{a_1} \sqcup \cdots \sqcup \exists \hat{R^k}_{a_k}$
$\mathsf{R\text{-}SET}^{-}_{\mathsf{Sub}}(\{R.a_1, R.a_2\}, \{S.b_1, S.b_2\})$	$\hat{R}_{a_1} \sqsubseteq \hat{S}_{b_1}$ where $a_1 \neq a_2, b_1 \neq b_2$
$\mathsf{R\text{-}SET}^{-}_{\mathsf{Exc}}(\{R.a_1, R.a_2\}, \{S.b_1, S.b_2\})$	$\hat{R}_{a_1} \sqsubseteq \neg \hat{S}_{b_1}$ where $a_1 \neq a_2, b_1 \neq b_2$
$\mathsf{R\text{-}SET}^{-}_{\mathsf{Sub}}(\{R.a\}, \{S.b\})$	$\exists \hat{R}_a \sqsubseteq \exists \hat{S}_b$
$\mathsf{R\text{-}SET}^{-}_{\mathsf{Exc}}(\{R.a\}, \{S.b\})$	$\exists \hat{R}_a \sqsubseteq \neg \exists \hat{S}_b$
$\mathsf{O\text{-}SET}_{\mathsf{Isa}}(O_1, \ldots, O_n, O)$	$O_1 \sqcup \cdots \sqcup O_n \sqsubseteq O$
$\mathsf{O\text{-}SET}_{\mathsf{Tot}}(O_1, \ldots, O_n, O)$	$O_1 \sqcup \cdots \sqcup O_n \sqsubseteq O$ $O \sqsubseteq O_1 \sqcup \cdots \sqcup O_n$
$\mathsf{O\text{-}SET}_{\mathsf{Ex}}(O_1, \ldots, O_n, O)$	$O_1 \sqcup \cdots \sqcup O_n \sqsubseteq O$ $O_i \sqsubseteq \sqcap_{j=i+1}^{n} \neg O_j$ for each $i = 1, \ldots, n$

Table 3.3: $\mathsf{ORM2}^{\mathsf{bin}}$ semantics

* A set $E_1, E_2, \ldots, E_n$ of concepts for entity types;
* a set $V_1, V_2, \ldots, V_m$ of concepts for value types;
* a set $D_1, D_2, \ldots, D_l$ of concepts for domain symbols;
* a set $R_1, R_2, \ldots, R_k$ of roles[2] for relation(predicate) symbols

This signature matches the signature of the linear syntax $\mathsf{ORM2}^{\mathsf{plus}}$. Besides the encoding of the $\mathsf{ORM2}^{\mathsf{plus}}$ constructs, we include three background axioms to simulate?? the UNA made by ORM2. These axioms are the following:

$$E_i \sqsubseteq \neg(D_1 \sqcup \cdots \sqcup D_l) \text{ for } i \in \{1, \ldots, n\} \tag{3.1}$$

$$V_i \sqsubseteq D_j \text{ for i } \in \{1, \ldots, m\}, \text{ and some } j \text{ with } 1 \leq j \leq l \tag{3.2}$$

$$D_i \sqsubseteq \sqcap_{j=i+1}^{l} \neg D_j \text{ for } i \in \{1, \ldots, l\} \tag{3.3}$$

Axiom (3.1) states that entity types are disjoint from domain symbols. That each value type is assigned a domain symbol is enforced by Axiom (3.2). The disjointness of the domain symbols results from Axiom (3.3).

Given an $\mathsf{ORM2}^{\mathsf{plus}}$ localised role $R.a$ we define the *directed relation* of R with respect to a, denoted $\hat{R}_a$, as

$$\hat{R}_a = \begin{cases} R & \text{if } a \text{ is the first ORM2 role of } R \\ R^{-} & \text{otherwise} \end{cases}$$

With this definition, we encode the $\mathsf{ORM2}^{\mathsf{bin}}$ constructs into DL as shown in Table 3.3 based on the FOL encoding for $\mathsf{ORM2}^{\mathsf{zero}}$ [8]. Given the DL constructs present in the encoding, our encoding belongs to the $\mathcal{ALCHIN}$ DL which extends the basic $\mathcal{ALC}$ DL with inverse roles ($\mathcal{I}$), role hierarchies ($\mathcal{H}$), and unqualified number restrictions($\mathcal{N}$).

[2] In this section, the term role will refer to DL roles unless otherwise specified.

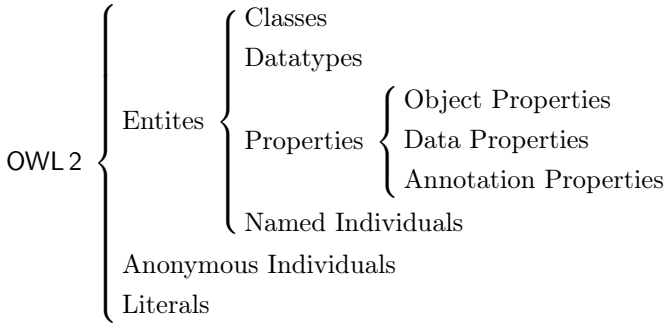


Figure 3.3: OWL 2 Elements

3.5 OWL 2

Ontologies are concrete descriptions of the world. OWL 2, the Web Ontology Language, is the standard Semantic Web language to define ontologies. It is based in the $\mathcal{SROIQ}^{(\mathcal{D})}$ DL, which besides the concept constructors of $\mathcal{ALC}$ allows for complex role inclusions ($\mathcal{R}$), nominals ($\mathcal{O}$), inverse roles ($\mathcal{I}$), qualified number restrictions ($\mathcal{Q}$), datatype properties ($^{(\mathrm{D})}$), and other additional features such local reflexivity of roles.

In this section we will describe the elements of OWL 2 using the functional-style syntax [28] and the direct semantics [29].

3.5.1 OWL 2 elements

Figure 3.3 shows a schema of the elements of OWL 2. OWL 2 entities, as we already mentioned, are classes, datatypes, properties, and named individuals. Classes are collections of individuals and datatypes collections of data values. Properties can be object properties, which interconnect individuals; or data properties, which connect individuals with literals; or annotation properties, which add nonlogical information to the ontology. The main difference between entities and non entities is that entities are uniquely identified by an IRI (Internationalized Resource Identifier) and non entities are not.

Anonymous individuals are individuals which exist but are not identified by an IRI [19]. For instance, in the example of Subsection 3.1.4 the person to which Jane is married is an anonymous individual. Although we ignore the name of this individual, we know this individual should exist; therefore, we can talk about it through its associations with other individuals.

Literals are the concrete values of the datatypes. Each literal has a lexical form, which is the string representing the value; and a datatype, which gives the meaning to the lexical form.

Two built-in classes are provided, owl:Thing, the set of all individuals; and owl:Nothing, the empty set. Analogous to the built-in classes, there are two built-in object properties: owl:topObjectProperty, which connects all possible pairs of individuals; and owl:bottomObjectProperty, which connects no individuals; and their respective built-in data properties owl:topDataProperty and

owl:bottomDataProperty.

Some examples of entities declaration using the OWL 2 functional-style syntax are shown in Listing 3.8, as well as the declaration of the build-ins. In the examples we create the atomic class `:Writer`, a named individual `:a` and a datatype `:ISBN`. We define the data property `:hasISBN` and the object property `:writes` as well.

```
1  Declaration(Class(:Writer))
2  Declaration(NamedIndividual(:a))
3  Declaration(Datatype(:ISBN))
4  Declaration(DataProperty(:hasISBN))
5  Declaration(ObjectProperty(:writes))
6
7  Declaration(Class(owl:Thing))
8  Declaration(Class(owl:Nothing))
9  Declaration(ObjectProperty(owl:topObjectProperty))
10 Declaration(ObjectProperty(owl:bottomObjectProperty
       ))
11 Declaration(DataProperty(owl:topDataProperty))
12 Declaration(DataProperty(owl:bottomDataProperty))
13 Declaration(Datatype(rdfs:Literal))
```

Listing 3.8: Functional-Style syntax OWL 2.

Besides these elements, OWL 2 provides expressions to create complex classes and properties. Complex data types are built through data ranges. One can One can state *axioms* about these elements, which are true statements of the domain.

The semantics of OWL 2 are the semantics of $\mathcal{SROIQ}^{(\mathcal{D})}$ DL. Therefore, we will structure the following introduction of the OWL 2 functional-style syntax based on its relation to DL. OWL 2 classes and datatypes are analogous to DL concepts up to some point. Class expression and data ranges correspond to DL concept constructors. Object properties and data properties are comparable to DL simple roles, while object and data property expressions correspond to DL role constructors. And OWL 2 individuals and literals are parallel to DL individuals.

First we will describe the available constructors in OWL 2. Then we will detail the terminological, relational, and assertional axioms. Finally, we will say some words regarding annotation properties and annotation assertions.

For the following expressions and axioms we adopt the this convention:

* `C` denotes a class and `C`* a class expression;
* `DT` denotes a datatype and `DR` a data range;
* `OP` denotes an object property and `OP`* an object property expression;
* `DP` denotes a data property and `DP`* a data property expression;
* α denotes an individual;
* ℓ denotes a literal; and
* `F` denotes a constraining facet.

3.5.2 OWL 2 constructors

3.5.2.1 Datatype restriction

Datatypes are sets of data values identified by an IRI. The base data values in OWL 2 are numbers, strings, booleans,time instants, IRIs, binary data, and XML literals. The datatypes for numbers include owl:real, owl:rational, xsd:decimal, xsd:interger, among others. For strings, the datatypes includes among other xsd:string, xsd:normalizedString, and xsd:language. *Facets* restrict datatypes, such as xsd:minInclusive, xsd:maxInclusive, xsd:minExclusive, and xsd:maxExclusive for numbers. For strings, the facets comprise xsd:length, xsd:minLength, xsd:maxLength, and xsd:pattern. A *data range*, is a set of tuples of literals. These tuples should be of the same size, which is the arity of the data range. Under this definition, datatypes are data ranges of arity 1.

The expression `DatatypeRestriction(DT` F_1 $\ell_1 \ldots \mathsf{F}_n$ ℓ_n`)` can constrain data ranges. In Listing 3.9 we define a new datatype for wine bottle capacities by restricting the valid strings using a regular expression.

```
1    DatatypeRestriction( xsd:string
2      xsd:pattern "(375|500|200|750)␣?ml
           |(1\.5|((1|3|5)(\.0)?))(␣|–)?(liter|l)"
3    )
```

Listing 3.9: `DatatypeRestriction` example

3.5.2.2 Concept intersection

To this constructor belong the class expression `ObjectIntersectionOf(`$\mathsf{C}_1^* \ldots \mathsf{C}_n^*$`)` and the data range operation `DataIntersectionOf(`$\mathsf{DR}_1 \ldots \mathsf{DR}_n$`)`. Examples of these expressions are in Listing 3.10, where the intersection of the classes `:Plant` and `:Carnivore` may be used to define a carnivore plant and the data range of person names such as "April", "May", and "June" is the intersection of `:monthName` and `:personName`. For `DataIntersectionOf`, the data ranges should have the same arity.

```
1  ObjectIntersectionOf(:Plant :Carnivore)
2  DataIntersectionOf(:monthName :personName)
```

Listing 3.10: `IntersectionOf` examples

3.5.2.3 Concept union

New concepts can also be created by concept union, in OWL 2 represented by `ObjectUnionOf(`$\mathsf{C}_1^* \ldots \mathsf{C}_n^*$`)` and `DataUnionOf(`$\mathsf{DR}_1 \ldots \mathsf{DR}_n$`)`. The individuals, or literals, defined by this restriction should belong to all the class expression, or data ranges.

3.5.2.4 Full negation

Class complement is expressed by `ObjectComplementOf(`C^*`)` and data range complement by `DataComplementOf(DR)`. Because of the OWA, for an individual, or data tuple, to be part of the set specified by this expression, it must be explicitly specified that it is not part of the class expression C^* , or the data range `DR`. With these two constructors one identifies everything that is not a `:Plant` or every data tuple which is not a `:monthName` as in Listing 3.11.

```
1 ObjectComplementOf(:Plant)
2 DataComplementOf(:monthName)
```

Listing 3.11: `ComplementOf` examples

3.5.2.5 Nominals

Nominals constructors are also found both for classes and data ranges. For classes OWL 2 provides `ObjectOneOf`($\alpha_1 \dots \alpha_n$) and for data ranges `DataOneOf`($\ell_1 \dots \ell_n$). The set defined by this restriction contains exactly the listed individuals, or literals. There are two more class expressions which use nominals: `DataHasValue(`$\mathtt{DP}^*$ ℓ`)` and `ObjectHasValue(`$\mathtt{OP}^*$ α`)`. The first expression outlines the class whose individuals are associated to ℓ by $\mathtt{DP}^*$. The class outlined by the second expression contains all the individuals related to α via $\mathtt{OP}^*$. Listing 3.12 shows Examples for these constructors.

```
1 ObjectOneOf(:guava :mango :papaya)
2 DataOneOf("May" "April" "June")
3 ObjectHasValue(:feedsOn :mango)
4 DataHasValue(:hasName "May")
```

Listing 3.12: `OneOf` and `HasValue` examples

3.5.2.6 Universal quantification

Given n data property expressions $\mathtt{DP}^*_1, \dots, \mathtt{DP}^*_n$ and a tuple of literals $(\ell_1, \dots, \ell_n)$, we say that an individual is *n-connected* by $\mathtt{DP}^*_1 \dots \mathtt{DP}^*_n$ to the tuple if for all $1 \leq i \leq n$ this individual is connected by $\mathtt{DP}^*_i$ to ℓ_i.

Because OWL 2 has two types of properties which give logical information, there are two universal quantification expressions. This also holds for existential quantification and the number restrictions.

A class can be universally quantified with `ObjectAllValuesFrom(`$\mathtt{OP}^*$ $\mathtt{C}^*$`)` or with the expression `DataAllValuesFrom(`$\mathtt{DP}^*_1 \dots \mathtt{DP}^*_n$ DR`)`. All individuals connected to individuals of this class by $\mathtt{OP}^*$ belong to the class defined by $\mathtt{C}^*$. And all tuples n-connected to individuals of this class by $\mathtt{DP}^*_1, \dots, \mathtt{DP}^*_n$ belong to the data range DR. In Listing 3.13 we have a model?? for the class of obligate carnivores, organisms which feed only on animal tissue. However, because of the OWA, this class also includes those individuals which do not feed at all. On line 2 we define a class whose individuals have as first name only month names. But again, by OWA this class also includes those individuals which do not have a first name.

```
1 ObjectAllValuesFrom(:feedsOn :AnimalTissue)
2 DataAllValuesFrom(:hasFirstName :monthName)
```

Listing 3.13: `AllValuesFrom` examples

3.5.2.7 Existential quantification

The existential quantification class expressions are `ObjectSomeValuesFrom(`$\mathtt{OP}^*$ $\mathtt{C}^*$`)` and `DataSomeValuesFrom(`$\mathtt{DP}^*_1 \dots \mathtt{DP}^*_n$ DR`)`. `ObjectSomeValuesFrom` defines a class which has all the individuals related at least once via $\mathtt{OP}^*$ to an individual of $\mathtt{C}^*$. For `ObjectSomeValuesFrom` the individuals which belong to this class are

n-connected by the data properties $\mathtt{DP}_1^*,\ldots,\mathtt{DP}_n^*$ to at least one tuple in DR. Listing 3.14 shows examples for these class expressions, the class of carnivores and omnivores; and the class of individuals with at least one month name as first name.

```
1 ObjectSomeValuesFrom( :feedsOn :AnimalTissue )
2 DataSomeValuesFrom( :hasFirstName :monthName )
```

Listing 3.14: SomeValuesFrom examples

3.5.2.8 Number restrictions

We group both qualified and unqualified number restrictions under qualified number restrictions, as unqualified number restrictions can be seen as been qualified by the top or bottom concept. In these definitions n is a non negative integer. The number restrictions using object properties are ObjectMinCardinality(n OP* [C*]), ObjectMaxCardinality(n OP* [C*]), and ObjectExactCardinality(n OP* [C*]). The analogous class expressions using data properties are DataMinCardinality(n DP* [DR]), DataMaxCardinality(n DP* [DR]), and DataExactCardinality(n DP* [DR]). For these expressions C* and DR are optional.

The class defined by an object cardinality expression has all the individuals related to at least, at most, or exactly n different individuals by OP*. If C* is also specified, then these n individuals should also belong to the class described by C*. For the data cardinality expression the class has those individuals connected by DP* to at least, at most, or exactly n different literals. If DR is defined, it should be a unary data range and the n literals should belong to it. The examples in Listing 3.15 outlines the class of individuals which have maximum three different first names, and the class of individuals married to exactly one person.

```
1 DataMaxCardinality(3 :hasFirstName)
2 ObjectExactCardinality(1 :isMarried :Person)
```

Listing 3.15: Cardinality examples

3.5.2.9 Inverse roles

In OWL 2 only object properties have inverse, as only entities?? can be in the subject position of a property. The inverse of an object property is obtained with the object property expression ObjectInverseOf(OP). For example, given the object property :hasAuthor the inverse property is equivalent to ObjectInverseOf(:hasAuthor).

3.5.2.10 Local reflexivity

The construct for local reflexivity or self restriction in OWL 2 is ObjectHasSelf(OP*) All the connected individuals to themselves by OP* belong to the restricted class. For instance, all the self-employed persons are defined by ObjectHasSelf(:employs).

3.5.3 Terminological axioms

3.5.3.1 Class expression axioms

Listing 3.16 shows an example for each of the class expression axioms. These axioms are used to express subclassing, class disjointnes and class equivalence.

```
SubClassOf(:Poet :Writer)
EquivalentClasses(:Person :HumanBeing)
DisjointClasses(:Person :Plant :Car)
DisjointUnion(:Person :Man :Woman)
```

Listing 3.16: Class expression axioms examples

Subclassing is stated with the axiom `SubClassOf(`C_1^* C_2^*`)`, which expresses that C_1^* is a subclass of C_2^*. `EquivalentClasses(`$C_1^* \dots C_n^*$`)` states that all the classes obtained by C_i^* have the same individuals. Class disjointness, a special form of subclassing where the superclass is $\bot$, is conveyed by the axiom `DisjointClasses(`$C_1^* \dots C_n^*$`)`, where for all $i, j, i \neq j$ the class C_i^* is disjoint with C_j^*. Finally, `DisjointUnion(C `$C_1^* \dots C_n^*$`)` states that the class `C` is equivalent to the union of C_1^*,...,C_n^* and these classes are disjoint.

3.5.3.2 Datatype definitions

`DatatypeDefinition(DT DR)`. This axiom states that the datatype `DT` is equivalent to the data range `DR`. Reusing the `DatatypeRestriction` example, we can define the datatype `:WineBottleCapacity` as shown in Listing 3.17.

```
Declaration(Datatype(:WineBottleCapacity))
DatatypeDefinition(:WineBottleCapacity
  DatatypeRestriction(xsd:string
    xsd:pattern "(375|500|200|750)␣?ml
        |(1\.5|((1|3|5)(\.0)?))(␣|–)?(liter|l)"
  )
)
```

Listing 3.17: `DatatypeDefinition` example

3.5.4 Relational axioms

These axioms express characteristics of the properties such as disjointness, subproperties, functional properties, reflexive properties, and more.

Subproperties, of object properties or data properties, are specified by the axioms `SubObjectPropertyOf(`OP_1^* OP_2^*`)` and `SubDataPropertyOf(`DP_1^* DP_2^*`)`. Any pair of individuals connected by DP_1^* are also connected by DP_2^*. And any pair of individual, literal connected by DP_1^* is also connected by DP_2^*. DP_2^* and DP_2^* may contain more pairs besides of the ones in its subproperty.

A special case of subproperty is the one defined by `SubObjectPropertyOf( ObjectPropertyChain(`$OP_1^* \dots OP_n^*$`) `OP^*`)`. The statement `ObjectPropertyChain(` $OP_1^* \dots OP_n^*$`)` forms a new property by the composition of the properties OP_1^*,...,OP_n^*. This new property is the one that is a subproperty of OP^*. Because literals cannot be in the subject position of a property, property chains are only allowed to be of object properties.

Formally,

$$\forall y_0, \dots, y_n : (y_0, y_1) \in (OP_1^*)^{OP} \wedge \dots \wedge (y_{n-1}, y_n) \in (OP_n^*)^{OP} \rightarrow (y_0, y_n) \in (OP^*)^{OP}$$

where $\cdot^{OP}$ is the part of the OWL 2 interpretation $\mathcal{I}$ called the *object property interpretation function* which assigns to each object property a subset of $\Delta^I \times \Delta^I$ [29]. Δ^I is the *object domain*, the set of all individuals.

If two or more properties are subproperties of each other then they have the same pairs of elements and they are equivalent. This can be stated with the axioms `EquivalentObjectProperties(`$\mathtt{OP}_1^* \ldots \mathtt{OP}_n^*$`)` or `EquivalentDataProperties(` $\mathtt{DP}_1^* \ldots \mathtt{DP}_n^*$`)`.

Properties, as classes, may also be disjoint, i.e., they share no pairs of elements. Axioms `DisjointObjectProperties(`$\mathtt{OP}_1^* \ldots \mathtt{OP}_n^*$`)` and `DisjointDataProperties(`$\mathtt{DP}_1^*$ $\ldots \mathtt{DP}_n^*$`)`.

Listing 3.18 shows examples of these axioms. The property formed by the property chain of properties `:isWineOfRegion` and `:locatedInCountry` is stated a subproperty of `isWineOfCountry`. This means that any wine which is of a region which is located in a country, is also a wine of this country. The properties `:wroteBook` and `:isAuthorOfBook` are equivalent, any individual who wrote a book is also the author of the same book.

```
1 SubObjectPropertyOf(
2   ObjectPropertyChain(:isWineOfRegion :
       locatedInCountry )
3   :isWineOfCountry)
4 EquivalentDataProperties(:wroteBook :isAuthorOfBook
    )
```

Listing 3.18: Subproperties and disjoint properties examples

That one property is the inverse of another is stated by `InverseObjectProperties` $(\mathtt{OP}_1^*\ \mathtt{OP}_2^*)$. This axiom conveys that $\mathtt{OP}_1^*$ is an inverse of the object property of $\mathtt{OP}_2^*$. For all properties `InverseObjectProperties(`$\mathtt{OP}^*$ `ObjectInverseOf(`$\mathtt{OP}^*$`))` also holds. An example of this axiom is `InverseObjectProperties(:isAuthorOfBook :bookWrittenBy)` where we state if someone is author of a book then this book is written by this someone.

Besides axioms that relate properties with other properties OWL 2 also has axioms that state characteristics of each property. We can specify the kind of individuals related with the property, the domain and range of the property. The domain of a property is specified by `ObjectPropertyDomain(`$\mathtt{OP}^*$ $\mathtt{C}^*$`)` for object properties and `DataPropertyDomain(`$\mathtt{DP}^*$ $\mathtt{C}^*$`)` for data properties. Ranges are specified with the corresponding axioms `ObjectPropertyRange(`$\mathtt{OP}^*$ $\mathtt{C}^*$`)` and `DataPropertyRange(`$\mathtt{DP}^*$ DR`)`. The axioms in Listing 3.19 state that the data property `:hasCapacity` has as domain the class `:Wine` and as range `:WineBottleCapacity`. That means that if something is connected with a datatype by `:hasCapacity`, this something is of class `:Wine` and the datatype is of the data range `:WineBottleCapacity`.

```
1 DataPropertyDomain(:hasCapacity :Wine)
2 DataPropertyRange(:hasCapacity :WineBottleCapacity)
```

Listing 3.19: Subproperties and disjoint properties examples

Other characteristics assigned to object property are functionality, inverse functionality, reflexivity, irreflexivity, symmetry, asymmetry, and transitivity. Only functionality can be stated regarding a data property. Listing 3.20 shows the signature of the axioms to state each of these characteristics.

```
1 FunctionalObjectProperty(OP*)
2 FunctionalDataProperty(DP*)
3 InverseFunctionalObjectProperty(OP*)
4 ReflexiveObjectProperty(OP*)
```

```
IrreflexiveObjectProperty(OP*)
SymmetricObjectProperty(OP*)
AsymmetricObjectProperty(OP*)
TransitiveObjectProperty(OP*)
```

Listing 3.20: Other properties axioms

3.5.4.1 Keys

OWL 2 also provide the axiom $\texttt{HasKey(C*}\ \texttt{(OP}_1^*\ldots\texttt{OP}_m^*\texttt{)}\ \texttt{(}\ \texttt{DP}_1^*\ldots\texttt{DP}_n^*\ \texttt{)}\ \texttt{)}$ which is similar to the database primary keys. Any two named individuals of class $\texttt{C}^*$ if they are connected by $\texttt{OP}_i^*$ to the same named individual α_i for each $1 \leq i \leq m$ and also are connected by $\texttt{DP}_j^*$ to the same literal ℓ_j for each $1 \leq j \leq n$ are the same individual. The direct semantics of this axiom are:

$$\begin{aligned}\forall x, y, z_1, \ldots, z_m, w_1, \ldots, w_n :\\ & x \in (\texttt{C}^*)^C \wedge x \in \texttt{NAMED} \wedge y \in (\texttt{C}^*)^C \wedge y \in \texttt{NAMED}\\ & \wedge \bigwedge_{i=1}^{m} \Big((x, z_i) \in (\texttt{OP}_i^*)^{\texttt{OP}} \wedge (y, z_i) \in (\texttt{OP}_i^*)^{\texttt{OP}} \wedge z_i \in \texttt{NAMED} \Big)\\ & \wedge \bigwedge_{j=1}^{n} \Big((x, w_j) \in (\texttt{DP}_j^*)^{\texttt{DP}} \wedge (y, w_j) \in (\texttt{DP}_j^*)^{\texttt{DP}} \Big)\\ & \rightarrow x = y\end{aligned}$$

where $\cdot^{\texttt{OP}}$ is the object property interpretation function, $\cdot^{\texttt{DP}}$ the *data property interpretation function*, which assigns to each data property a subset of $\Delta^I \times \Delta^D$, and NAMEDis the subset of Δ^I which has all *named* individuals [29]. Δ^D is the *data domain*, the set of all literals.

3.5.5 Assertional axioms

OWL 2 assertions are the parallel of DL assertional axioms. These axioms allow us to state facts about individuals. Two or more individuals are declared equal with the assertion $\texttt{SameIndividual(}\alpha_1 \ldots \alpha_n\texttt{)}$. The counterpart of this axiom is $\texttt{DifferentIndividuals(}\alpha_1 \ldots \alpha_n\texttt{)}$.

$\texttt{ClassAssertion(C*}\ \alpha\texttt{)}$ specifies that the individual α belongs to class $\texttt{C}^*$. Similarly, assertions $\texttt{ObjectPropertyAssertion(OP*}\ \alpha_1\ \alpha_2\texttt{)}$ and $\texttt{DataPropertyAssertion (DP*}\ \alpha\ \ell\texttt{)}$ specify that the individual α_1 is related to α_2 by $\texttt{OP}^*$ and that the individual α is related to ℓ by $\texttt{DP}^*$. The negative form of the last two assertions are $\texttt{NegativeObjectPropertyAssertion(OP*}\ \alpha_1\ \alpha_2\texttt{)}$ and $\texttt{NegativeDataPropertyAssertion (DP*}\ \alpha\ \ell\texttt{)}$, respectively.

Listing 3.21 shows examples of these assertions. First we state that `:mellvile` and `:mobydick` are different individuals. Then than `:mobydick` is of class `:Book` and that `:mellvile` wrote the book `:mobydick`. Finally we declare that the title of `:mobydick` is not `"Moby_Dick"`.

```
DifferentIndividuals(:melville :mobydick)
ClassAssertion(:Book :mobydick)
ObjectPropertyAssertion(:wrote :melville :mobydick)
```

```
4 NegativeDataPropertyAssertion(:hasTitle :mobydick
      "Moby␣Dick")
```

Listing 3.21: Assertion examples

3.5.6 Annotations

Finally the OWL 2 functional-style syntax allows us to declare non logical information about the ontology, axioms and entities with annotations properties. The annotation properties can have a domain, a range, and subproperties just as data and object properties. Built-in annotation properties include rdfs:label and rdfs:comment.

Entities or anonymous individuals are annotated using annotation assertions `AnnotationAssertion(AP AS AV )` where `AP` is an annotation property; `AS` is the annotation subject, an entity or an anonymous individual; and `AV` is the value of the annotation, an anonymous individual, an IRI, or a literal.

Axioms can also be annotated. Actually, all axioms already introduced have an initial optional parameter: a list of annotations. These annotations are stated by `Annotation(AP AV)`

Listings 3.22 shows examples of annotations. We annotate the individual `:melville` with a label to give it a descriptive name `"Herman␣Mellvile"`. The `ObjectPropertyChain` axiom of Listing 3.18 is now annotated with a comment to give a verbal explanation.

```
1 AnnotationAssertion(rdfs:label :mellvile "Herman
     Mellvile")
2 SubObjectPropertyOf(
3   Annotation(rdfs:comment "States that if a wine is
       of a region which is located in a country,
      then the wine is of this country as well." )
4   ObjectPropertyChain(:isWineOfRegion :
      locatedInCountry) :isWineOfCountry)
5 )
```

Listing 3.22: Annotations

3.6 Mapping ORM2^{bin} to OWL 2

Now that we have all the background knowledge and the formal semantics of both ORM2^{bin} and OWL 2, we can do the mapping. We based the mapping in the OWL 2 Direct Semantics [29] and in the OWL 2 Structural Specification and Functional-Style Syntax [28].

3.6.1 Entities declaration

The ORM2^{bin} syntax includes the declaration of the entity types, value types and relations. Entity types are mapped to OWL 2 classes and value types to datatypes. Relations correspond to properties, but first it is necessary to differentiate between the relations which connect two entity types, mapped to object properties, and the relations which connect an entity type with a value type, mapped to data properties. We do not consider the relations which connect two value types, as OWL 2 does not allow data values to be in the subject position

ORM2^{bin} Construct	DL Semantics	OWL 2 Axiom
TYPE$(R.1, E)$	$\exists R \sqsubseteq E$	`ObjectPropertyDomain( R E )`
TYPE$(R.2, E)$	$\exists R^- \sqsubseteq E$	`ObjectPropertyRange( R E )`
TYPE$(D.1, E)$	$\exists D \sqsubseteq E$	`DataPropertyDomain( D E )`
TYPE$(D.2, V)$	$\exists D^- \sqsubseteq V$	`DataPropertyRange( D V )`

Table 3.4: TYPE mapping

of properties. If there is any relation connecting a value type with an entity type, in this order, it is inverted by providing an alternate inverse predicate reading and updating the diagram accordingly. The mapping of the entities in Listing 3.23 is shown in Listing 3.24.

```
1 ENTITYTYPES: {WineProduct, Appellation}
2 VALUETYPES: {Year}
3 RELATIONS: {belongsToAppellation, hasVintage}
```

Listing 3.23: ORM2^{bin} declarations

```
1 Declaration(Class(:WineProduct))
2 Declaration(Class(:Appellation))
3 Declaration(Datatype(:Year))
4 Declaration(ObjectProperty(:belongsToAppellation))
5 Declaration(DataProperty(:hasVintage))
```

Listing 3.24: OWL 2 declarations

3.6.2 TYPE

To translate the construct TYPE$(R.a, O)$ to OWL 2 we have to consider the four cases shown in Figure 3.4.

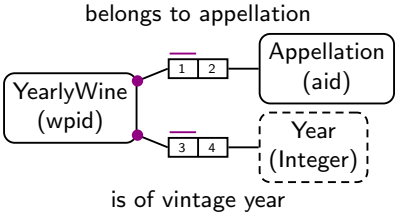


Figure 3.4: ORM2 diagram with the four TYPE cases

1. R is mapped to an object property and a is its first role
2. R is mapped to an object property and a is its second role
3. R is mapped to a data property and a is its first role
4. R is mapped to a data property and a is the second role

The semantics and mapping to OWL 2 of each of these cases are shown in Table 3.4.

Listing 3.25 shows the TYPE constructs of the diagram in Figure 3.4, and Listing 3.26 the mapping into OWL 2.

ORM2^{bin} Syntax	DL Semantics	OWL 2 Axiom
FREQ^{-}(R.1,$\langle n,m\rangle$)	$\exists R \sqsubseteq \geq n\, R \sqcap \leq m\, R$	`ObjectPropertyDomain(R ObjectIntersectionOf( ObjectMinCardinality(n R) ObjectMaxCardinality(m R)))`
FREQ^{-}(R.2,$\langle n,m\rangle$)	$\exists R^{-} \sqsubseteq \geq n\, R^{-} \sqcap \leq m\, R^{-}$	`ObjectPropertyRange(R ObjectIntersectionOf ( ObjectMinCardinality(n R) ObjectMaxCardinality(m R)))`
FREQ^{-}(D.1,$\langle n,m\rangle$)	$\exists D \sqsubseteq \geq n\, D \sqcap \leq m\, D$	`DataPropertyDomain(R DataIntersectionOf( DataMinCardinality(n R) DataMaxCardinality(m R)))`
FREQ^{-}(D.2,$\langle n,m\rangle$)	$\exists D^{-} \sqsubseteq \geq n\, D^{-} \sqcap \leq m\, D^{-}$	Not mappable

Table 3.5: FREQ^{-}mapping

```
TYPE(belongsToAppellation.1, YearlyWine)
TYPE(belongsToAppellation.2, Appellation)
TYPE(isOfVintageYear.1, YearlyWine)
TYPE(isOfVintageYear.2, Year)
```

Listing 3.25: ORM2^{bin} TYPE cases

```
ObjectPropertyDomain(:belongsToAppellation :
    WineVintage)
ObjectPropertyRange(:belongsToAppellation :
    Appellation)
DataPropertyRange(:isOfVintage :Year)
DataPropertyDomain(:isOfVintage :YearlyWine)
```

Listing 3.26: OWL 2 TYPE cases

3.6.3 FREQ^{-}

To map the FREQ^{-} construct into OWL 2 we consider the same four cases as in the mapping of TYPE. n is the minimum value and m the maximum.

The last case is not mappable as OWL 2 literals are not OWL 2 entities and are not allowed in the subject position of a property. If $m = n$, the intersection of cardinality expressions can be replaced by an `ObjectExactCardinality(n R)` or `DataExactCardinality(n D)`, respectively.

3.6.4 MAND

The MAND construct mapping is shown in Table 3.6. Each directed relation $\hat{R^i}_{a_i}$, is mapped as follows.

$$\hat{R^i}_{a_i} \text{ is mapped to } \begin{cases} R^i & \text{if } a_1 \text{ is the first ORM2 role of } R^i \\ \texttt{ObjectInverseOf}(R^i) & \text{otherwise} \end{cases}$$

ORM2$^{\text{bin}}$ Syntax	DL Semantics	OWL 2 Axiom
MAND({ $R^1.a_1,\ldots,$ $R^k.a_k, O$ })	$O \sqsubseteq \exists \hat{R^1}_{a_1} \sqcup \cdots \sqcup \exists \hat{R^k}_{a_k}$	SubClassOf(O ObjectUnionOf(RMinCardinality$_1$(1 $\hat{R^1}_{a_1}$) ... RMinCardinality$_k$(1 $\hat{R^k}_{a_k}$)))

Table 3.6: MAND mapping

ORM2$^{\text{bin}}$ Syntax	DL Semantics	OWL 2 Axiom
R-SET$^{-}_{\text{Sub}}$($\{R^1.a_{11}, R^1.a_{12}\}$, $\{R^2.a_{21}, R^2.a_{22}\}$)	$\hat{R^1}_{a_{11}} \sqsubseteq \hat{R^2}_{a_{21}}$ $a_{11} \neq a_{12}, a_{21} \neq a_{22}$	RSubPropertyOf($\hat{R^1}_{a_{11}}$ $\hat{R^2}_{a_{21}}$)
R-SET$^{-}_{\text{Exc}}$($\{R^1.a_{11}, R^1.a_{12}\}$, $\{R^2.a_{21}, R^2.a_{22}\}$)	$\hat{R^1}_{a_{11}} \sqsubseteq \neg\hat{R^2}_{a_{21}}$ $a_{11} \neq a_{12}, a_{21} \neq a_{22}$	RDisjointProperties($\hat{R^1}_{a_{11}}$ $\hat{R^2}_{a_{21}}$)

Table 3.7: R-SET$_{\text{H}}$ for the whole relation mapping

Only object properties have inverse in OWL 2; therefore, if R^i is a data property and a_1 is the second role of R^i, the set of pairs defined by $\hat{R^i}_{a_i}$ is not definable in OWL 2. For this constructor and the following we derive all the possible cases and outline which ones are mappable to owl.

For this constructor, we add the restriction that for all $\hat{R^i}_{a_i}$ is R^i is a data property, then a_i must be the first role of the relations. Otherwise we remove this constraint. If R^i is an object property, RMinCardinality is ObjectMinCardinality otherwise it is DataMinCardinality.

3.6.5 R-SET$^{-}_{\text{H}}$

Tables 3.7 and 3.9 show the mappings for the construct R-SET$^{-}_{\text{H}}$ in its two flavours. If R^i is an object property, RSubPropertyOf is SubObjectPropertyOf and RDisjointProperties is DisjointObjectProperties. Otherwise RSubPropertyOf is SubDataPropertyOf and RDisjointProperties is DisjointDataProperties. We consider eight cases derived by the kind of property of R^i (two cases) and the index of a_{i1} (four cases) which are shown in Table 3.8.

Two cases are removed because literals cannot be compared with entities. Although there is no inverse for data properties, the corner case when both relations are data properties and for both a_{i1} is the second role of the relation, can be modelled because we are considering the whole relation. In this case $\hat{R^1}_{a_{11}} \sqsubseteq \hat{R^2}_{a_{21}}$ is equivalent to $\hat{R^1}_{a_{12}} \sqsubseteq \hat{R^2}_{a_{22}}$ and there is no need to obtain the inverse of the data properties.

If R^i is an object property, RSomeValuesFrom$_i$ is ObjectSomeValuesFrom and rTop$_i$ is owl:Thing. Otherwise RSomeValuesFrom$_i$ is DataSomeValuesFrom and rTop$_i$ is rdfs:Literal.

There are sixteen possible cases depending on the kind of property of R^i (four cases) and the index of a_i (four cases). All these cases are shown in Table 3.10. Six of these cases are ignored because literals cannot be compared with entities. One last case remains unknown, when both R^i are data properties and the a_i's

	R^1, R^2 are object properties		R^1, R^2 are data properties	
Cases	$\hat{R^1}_{a_{11}} = R^1$	$\hat{R^1}_{a_{11}} = R^{1^-}$	$\hat{R^1}_{a_{11}} = R^1$	$\hat{R^1}_{a_{11}} = R^{1^-}$
$\hat{R^2}_{a_{21}} = R^2$	OK	OK	OK	X
$\hat{R^2}_{a_{21}} = R^{2^-}$	OK	OK	X	*OK*

Table 3.8: Cases for R-SET$^{-}_{H}$ for the whole relation

ORM2^{bin} Syntax	DL Semantics	OWL 2 Axiom
R-SET$^{-}_{Sub}$($\{R^1.a_1\}, \{R^2.a_2\}$)	$\exists\hat{R^1}_{a_1} \sqsubseteq \exists\hat{R^2}_{a_2}$	`SubClassOf( RSomeValuesFrom`$_1$`(` $\hat{R^1}_{a_1}$ `: rTop`$_1$`) RSomeValuesFrom`$_2$`(`$\hat{R^2}_{a_2}$ `:rTop`$_2$`))`
R-SET$^{-}_{Exc}$($\{R^1.a_1\}, \{R^2.a_2\}$)	$\exists\hat{R^1}_{a_1} \sqsubseteq \neg\exists\hat{R^2}_{a_2}$	`DisjointClasses(RSomeValuesFrom`$_1$`(` $\hat{R^1}_{a_1}$ `:rTop`$_1$`) RSomeValuesFrom`$_2$`(`$\hat{R^2}_{a_2}$ `:rTop`$_2$`) )`

Table 3.9: R-SET$^{-}_{H}$ for just one role mapping

are the second role of the relation. This case could be modelled using data ranges; however, currently we ignore it in our mapping.

3.6.6 O-SET$_{H}$

ORM2 subtyping is performed in *object types* [18], but in OWL 2 only offers subclasses. Therefore, for the following, O is an entity type mapped to a class. Table 3.11 shows the mapping for the three subtype constructs O-SET$_{Isa}$,O-SET$_{Tot}$,O-SET$_{Ex}$, and for the partition subtyping which is the union of O-SET$_{Tot}$ and O-SET$_{Ex}$.

3.6.7 PRIMARY and LABEL

The ORM2^{bin} syntax for the diagram in Figure 3.5 shown in Listing 3.27. The direct translation to OWL 2 using the mapping just introduced is in Listing 3.28. However, the actual information we use in the ontology for these diagram segment is shown in Listing 3.29.

ENTITYTYPES: `{Region}`

		R^1 is an object property		R^1 is a data property	
Cases		$\hat{R^1}_{a_1} = R^1$	$\hat{R^1}_{a_1} = R^{1^-}$	$\hat{R^1}_{a_1} = R^1$	$\hat{R^1}_{a_1} = R^{1^-}$
R^2 is an object property	$\hat{R^2}_{a_2} = R^2$	OK	OK	OK	X
	$\hat{R^2}_{a_2} = R^{2^-}$	OK	OK	OK	X
R^2 is a data property	$\hat{S^2}_{a_2} = R^2$	OK	OK	OK	X
	$\hat{R^2}_{a_2} = R^{2^-}$	X	X	X	?

Table 3.10: Cases for R-SET$^{-}_{H}$ for just one role

ORM2$^{\text{bin}}$ Syntax	DL Semantics	OWL 2 Axiom
O-SET$_{\text{Isa}}(O_1,\ldots,O_n,O)$	$O_1 \sqcup \cdots \sqcup O_n \sqsubseteq O$	`SubClassOf(ObjectUnionOf(O1 ... On) O)`
O-SET$_{\text{Tot}}(O_1,\ldots,O_n,O)$	$O_1 \sqcup \cdots \sqcup O_n \sqsubseteq O$ $O \sqsubseteq O_1 \sqcup \cdots \sqcup O_n$	`EquivalentClasses( ObjectUnionOf(O1 ... On) O)`
O-SET$_{\text{Ex}}(O_1,\ldots,O_n,O)$	$O_1 \sqcup \cdots \sqcup O_n \sqsubseteq O$ $O_i \sqsubseteq \sqcap_{j=i+1}^{n} \neg O_j$ for each $i = 1,\ldots,n$	`SubClassOf(ObjectUnionOf(O1 ... On) O) DisjointClasses(O1 ... On)`
O-SET$_{\text{Ex}}(O_1,\ldots,O_n,O)$ O-SET$_{\text{Tot}}(O_1,\ldots,O_n,O)$	$O_1 \sqcup \cdots \sqcup O_n \sqsubseteq O$ $O \sqsubseteq O_1 \sqcup \cdots \sqcup O_n$ $O_i \sqsubseteq \sqcap_{j=i+1}^{n} \neg O_j$ for each $i = 1,\ldots,n$	`DisjointUnion(O O1 ... On)`

Table 3.11: O-SET$_{\text{H}}$mapping

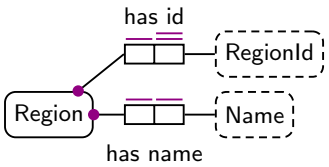


Figure 3.5: Extended *Region* diagram

```
VALUETYPES: {RegionName,RegionId}
RELATIONS: {hasRegionId, hasRegionName}

TYPE(hasRegionId.1, Region)
TYPE(hasRegionId.2, RegionId)
TYPE(hasRegionName.1, Region)
TYPE(hasRegionName.2, RegionName)

FREQ–(hasRegionId.1, (0,1))
FREQ–(hasRegionId.2, (0,1))
FREQ–(hasRegionName.1, (0,1))
FREQ–(hasRegionName.2, (0,1))

MAND({ hasRegionId.1}, Region)
MAND({ hasRegionName.1}, Region)

PRIMARY(Region hasRegionId.2)
LABEL(Region, hasRegionName.2)
```

Listing 3.27: ORM2$^{\text{bin}}$ example

```
Declaration(Class(:Region))
Declaration(Datatype(:RegionId))
DatatypeDefinition(:RegionId xsd:integer)
Declaration(Datatype(:RegionName))
DatatypeDefinition(:RegionName xsd:string)
```

Regions	
Region id	Region name
10039	France - Rhône
105	Italy
109	Spain

Table 3.12: Region data

```
Declaration(DataProperty(:hasRegionId))
Declaration(DataProperty(:hasRegionName))

DataPropertyDomain(:hasRegionId :Region)
DataPropertyRange(:hasRegionId :RegionId)
DataPropertyDomain(:hasRegionName :Region)
DataPropertyRange(:hasRegionName :RegionName)

DataPropertyDomain(:hasRegionId DataIntersectionOf(
     DataMinCardinality(0 :hasRegionId)
    DataMaxCardinality(1 :hasRegionId)))
% FREQ-(hasRegionId.2, (0,1)) not mapped
DataPropertyDomain(:hasRegionName
    DataIntersectionOf( DataMinCardinality(0 :
    hasRegionName) DataMaxCardinality(1 :
    hasRegionName)))
% FREQ-(hasRegionName.2, (0,1)) not mapped

SubClassOf(:Region  DataMinCardinality(1 :
    hasRegionName ))
SubClassOf(:Region  DataMinCardinality(1 :
    hasRegionName ))
```

Listing 3.28: OWL 2 mapped example

```
% Actual OWL 2 ontology for Region
Declaration(Class(:Region))
```

Listing 3.29: Actual OWL 2 ontology used

The relations linked to PRIMARY and LABEL in ORM2bin are not mapped to their corresponding OWL 2 declarations. Instead, for each individual we use the value of the role in PRIMARY as part of the individual's IRI and the value of the role in LABEL as the rdfs:label. We discard the properties :hasRegionId and :hasRegionName as these concepts are not useful for Quelo, but the data linked to these relations will be used in the individual assertions. Given the data[3] in Table 3.12, to add this data to the ontology in Listing 3.28 we would add the declarations and assertions of Listing 3.30. In the actual ontology; however, we add the ones shown in Listing 3.31. The number of assertions is lower and for our purposes both provide the same data. Although in the second case we know which is the preferred identifier and label for the class, and use them accordingly.

```
Declaration(NamedIndividual(:reg10039))
Declaration(NamedIndividual(:reg105))
```

[3]Data obtained from Wine.com

```
Declaration(NamedIndividual(:reg109))
DataPropertyAssertion(:hasRegionId :reg10039 "10039
    "^^xsd:integer)
DataPropertyAssertion(:hasRegionName :reg10039 "
    France␣–␣Rhône"^^xsd:string)
DataPropertyAssertion(:hasRegionId :reg10039 "105"
    ^^xsd:integer)
DataPropertyAssertion(:hasRegionName :reg10039 "
    Italy"^^xsd:string)
DataPropertyAssertion(:hasRegionId :reg10039 "109"
    ^^xsd:integer)
DataPropertyAssertion(:hasRegionName :reg10039 "
    Spain"^^xsd:string)
```

Listing 3.30: OWL 2 assertions example

```
Declaration(NamedIndividual(:reg10039))
Declaration(NamedIndividual(:reg105))
Declaration(NamedIndividual(:reg109))
AnnotationAssertion(rdfs:label :reg10039 "France␣–␣
    Rhône")
AnnotationAssertion(rdfs:label :reg105 "Italy")
AnnotationAssertion(rdfs:label :reg109 "Spain")
```

Listing 3.31: OWL 2 assertions and annotations example

We include the taxonomy of the resulting ontology in Appendix E.

Chapter 4

Quelo: Querying Data in Ontologies

In this chapter we will introduce Quelo, Section 4.1, and some of its extensions and future work, Section 4.2. In Section 4.3 we explain the validation process and our results.

4.1 Quelo

Disclaimer: most of the description of Quelo is from a previous report [23].

Quelo is a Natural Language Interface for ontologies which attempts to cover the gap between the information stored in ontologies and the lay user [9]. The main idea is to allow the user to construct a natural language query according to an underlying ontology. The framework guides the user with menus to create a query which is consistent with the ontology. Because Quelo is a guided framework the user does not have to have any previous knowledge about the ontology structure or the language used to execute the query. The text shown to the user is automatically generated from the formal query language of the ontology with a Natural Language Generation (NLG) process based on templates. An example of such a query is shown in Figure 4.1.

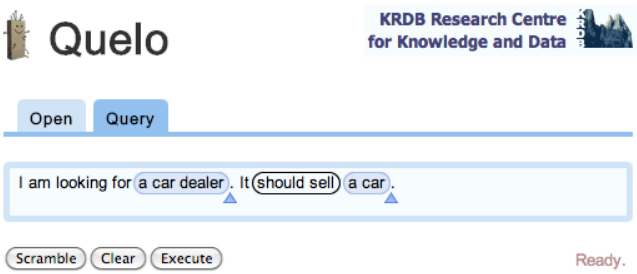


Figure 4.1: Quelo query example

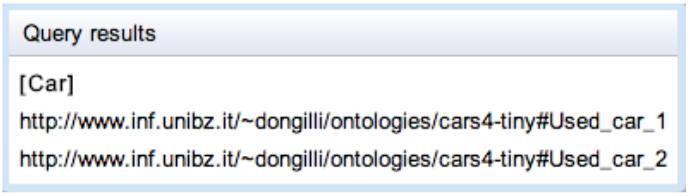


Figure 4.2: Quelo results table

The query changes according to the choices of the user showing the verbalisation of the query in English. Given a query the user can extend it by adding compatible concepts or new properties for an existing concept. The user can also replace part of the query, or all of it, with a related concept; either a super-concept, an equivalent concept, or a sub-concept. It is also possible to delete a selection of the query. The user may also select one or more concepts as the projected concepts for the query [35].

The current implementation of Quelo shows the results of the query, a list of IRIs, in a modal window as a table, where each of the columns has as the label of the concept as header. The answers for the query in Figure 4.1 can be seen in Figure 4.2.

New extensions under development give better verbalisations based in Tree-Adjoining Grammars [34] and provide descriptive headers for the results table [23].

The next step is to include data values in the query. But any progress in this direction requires an ontology with data compatible with Quelo's query tool. With an ontology with real world data these new features can be tested and validated.

4.2 Quelo extensions

The current version of Quelo only queries the TBox of the ontology. The answers given are from the assertions in the ABox, but there is no possibility to use the values in the ABox to constraint the query. To test any further extension of Quelo we need an ontology with data values.

Using the new ontology, for example, we can create a better results table, as shown in Figure 4.3. Instead of just presenting the list of IRIs we provide links to the description web page of each *Appellation* provided by Wine.com. The rdfs:label value is used as the link name, and it gives a descriptive name to each entry. The header helps the user know to which concepts all these names belong. We are exploring other extensions to use the data already obtained such as the one proposed by Ngo [30].

4.3 Ontology validation

To validate the ontology we created from the methodology and mapping just described, we use Quelo *Scramble* feature. This feature creates a random query from the ontology using the allowed operations of Quelo query tool and outputs the verbalisation for each of them.

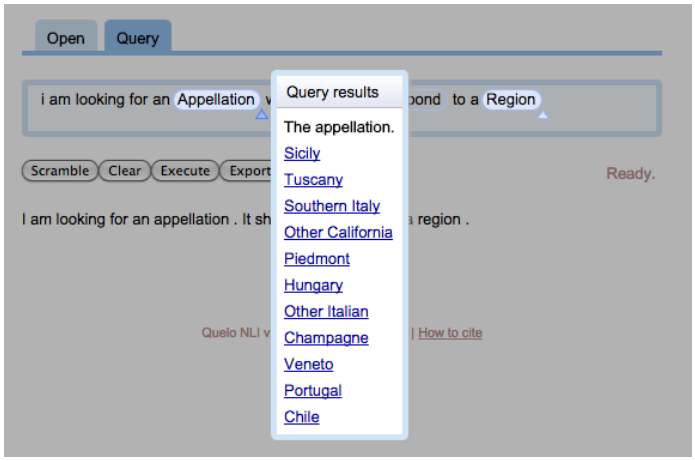


Figure 4.3: Quelo appellations results

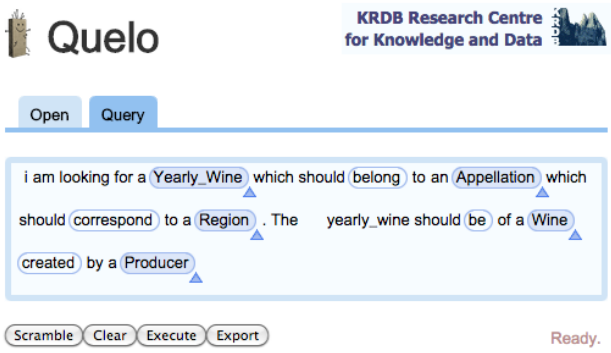


Figure 4.4: Quelo wine ontology query example

If an ontology is not correctly specified, Quelo will create valid queries according to the specification, but incorrect according to the domain the ontology is modelling. The generated verbalisations are queries any user can create from the ontology using Quelo. If the query does not reflect the domain of the ontology this defeats the purpose of the ontology specification.

We generated 36 random queries from our ontology, of which Quelo produced 176 unique verbalizations. Quelo produces several verbalisations per query, but only displays the best one according to its ranking system. Of these verbalisations all the best ones, the ones displayed, were correct with respect to the domain modelled.

Some of the queries included up to seven different concepts, but the verbalisations were coherent. For example, the query shown in Figure 4.4 spans over five concepts of the ontology and the verbalisation is fluid and unambiguous.

Some verbalisation improvements are still possible. Quelo creates a lexicon for

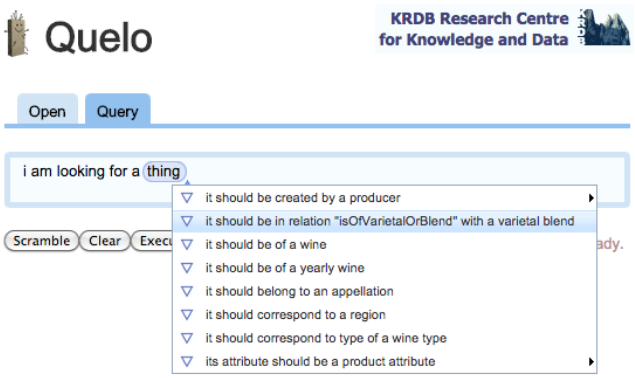


Figure 4.5: Quelo generic relation

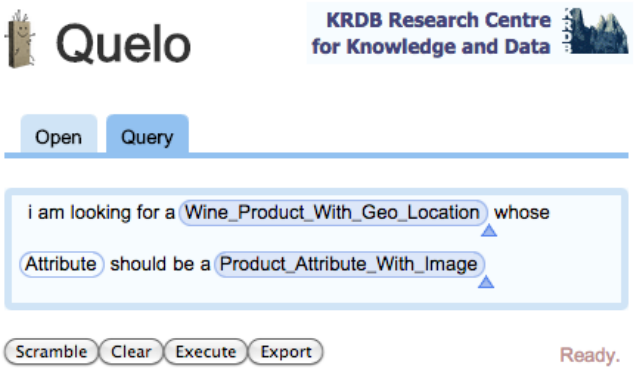


Figure 4.6: Quelo determiners discrepancy

classes and properties based on the fragment of the IRI and its annotations. One of the relations was not classified in the lexicon; and therefore, no verbalisations included this relation. This relation was "isOfVarietalOrBlend " and, as shown in Figure 4.5, Quelo treats it as a generic string in the menus. But because there is no entry for it in the lexicon, no verbalisation are produced with it.

Another area of improvement are the determiners for many to many relations. In the ontology *Wine Product* and *Product Attribute* are associate with a many to many relation hasAttribute. However, in the verbalisation of Figure 4.6 the pronoun used implies that there is only one *Product Attribute* assigned to each *Wine Product*. A better verbalisation is "I am looking for a wine product with geo location *one of whose attributes* should be . . . "

We also produced 33 verbalisations from 13 random queries with the ontology, but keeping the hasName relations. Adding these relations produced verbalisations as the following

∗ I am looking for a type name.

* I am looking for a thing whose wine type name should be a type name.
* I am looking for a wine type whose wine type name should be a type name.

These queries hinder the exploration of the ontology. The only information we obtain is that a *Wine Type* has a *Type Name*, which is expected. Everything should have a name.

By removing these relations from the ontology, we remove this sort of queries. With all the remaining queries new concepts and relations of the ontology may be discovered. Therefore, removing these relations had a positive impact in the exploration of the ontology.

Chapter 5

Conclusions

Data is useful as long as it is accessible. Ontologies remove the need to know the whole structure of the data before querying. And tools like Quelo remove the need to know the query language. So data in ontologies is fully accessible and desirable.

In this thesis we described a methodology to port raw data into an ontology. With this methodology any unstructured data can be made accessible to any user. We also presented a mapping from a well defined subset of ORM2 into OWL 2.

The ontology created is valid according to the specifications of Quelo for which it was created. Therefore, we can use this ontology as a data source to test new extensions of Quelo, particularly the ones which will include attributes and constants to the query.

Although ORM2 and OWL 2 come from different perspectives, they can be used together to complement each other. ORM2 diagrams are useful to communicate with the domain expert or just to visualize the concept model in an concise form. ORM2 provides with a wide range of constructs to represent data constraints. OWL 2 gives all the possibilities of the Semantic Web. Data in ontologies is reachable to anyone in the Web. One can also reason on the ontology and extract inferences from the knowledge represented. The mapping created in this thesis allows us to obtain the benefits of both representations.

5.1 Future work

Further development could be made to expand $\mathsf{ORM2}^{\mathsf{bin}}$ to cover other common constructs of ORM2 such as value restrictions. However, to make a formal mapping, first the construct should be given a DL semantics. The FOL and set semantics for this and other constructs exists, but the encoding into DL, more specifically $\mathcal{SROIQ}^{(\mathcal{D})}$ DL which is the logic used by OWL 2, is lacking. Once the DL semantics for these constructs is formulated a mapping to the OWL 2 direct semantics can be done.

Most of the cases involving a datatype as the second role of the relation where ignored in the present mapping. We could explore reformulations of these constraints using OWL 2 data range expressions. As Quelo only creates

queries regarding the terminology of the ontology, this omissions do not impact the validation of the ontology using Quelo. However, these constraints may be relevant to other applications and therefore they should be included in the ontology if they were present in the conceptual model.

Appendices

Appendix A

Properties Descriptions

We changed the names of some of the properties and objects to remove ambiguous terms. For example, we changed Year to VintageYear and *Varietal* to *VarietalBlend*. The object *Vineyard* actually refers to the wine producer, and we changed the name to *Producer* (See Table A.1). For the construction of the conceptual schema we ignored some properties, like the URLs to other products. We also removed the redundant the property Type. All the data was obtained from Wine.com.

Prod Id	**Product Name**	**"Vineyard" [Id]**
79426	Massolino *Vigna Margheria* Barolo 2000	Massolino[5244]
129221	Massolino *Vigna Margheria* Barolo 2007	Massolino[5244]
130527	Massolino *Vigna Margheria* Barolo 2009	Massolino[5244]
78390	Massolino *Vigna Parafada* Barolo 2000	Massolino[5244]
130528	Massolino *Vigna Parafada* Barolo 2009	Massolino[5244]
130532	Massolino *Vigna Parussi* Barolo 2009	Massolino[5244]
91543	Massolino Barolo *Vigna Rionda* Riserva 2000	Massolino[5244]
130536	Massolino *Vigna Rionda* Riserva Barolo 2007	Massolino[5244]

Table A.1: Vineyards are producers.

Property	Abbrv.	Description
WineProduct_Id	WPID	A unique identifier for the wine product.
Name	WPN	The product name.
GeoLocation_Url	WPG	The URL to a map of the wine.
Url	WPU	The URL to the product detail web page.
PriceMin	WPM	The starting point for the price.
PriceMax	WPX	The maximum price point across all markets for this product.
PriceRetail	WPR	The suggested retail price for the product.
Ratings_HighestScore	WPT	The highest score given to the product. Ex: 0,1,..,100
Reviews_HighestScore	WPV	The highest review score given to the product. Ex: 0,1,..,5
Type	WPT	Identifies the type of product that is being described. For wines is always 'Wine'
VintageYear	WPY	The vintage year of the product or 'Non-Vintage'. Can be null. Ex: 2013, 2014
Appellation_Id	AID	A unique identifier for the appellation to which the product belongs.
Appellation_Name	AN	The name of the appellation. Ex: Veneto, Tuscany
Appellation_Url	AU	The URL to the other products in the same appellation.
Region_Id	RID	A unique identifier for the region that corresponds to the appellation.
Region_Name	RN	The name of the region. Ex: Italy, France - Bordeaux
Region_Url	RU	The URL to the other products in the same region.
VarietalBlend_Id	VBID	A unique identifier for the varietal that the product belongs to.
VarietalBlend_Name	VBN	The name of the varietal. Ex: Merlot, Bordeaux Red Blends, Other Red Blends
VarietalBlend_Url	VBU	The URL to other products that have the same varietal.
WineType_Id	TID	A unique identifier for the type of wine that the varietal corresponds to.
WineType_Name	TN	The name of the type of wine. Ex: White Wines, Red Wines, Champagne & Sparkling
WineType_Url	TU	The URL to other products that have the same wine type.
Producer_Id	PID	A unique identifier for the producer that creates the product.
Producer_Name	PN	The name of the producer. Ex: Moet & Chandon, Frescobaldi
Producer_Url	PU	The URL to the description of the producer.
Producer_ImageUrl	PIU	The URL to an image of the producer.
Producer_GeoLocation_Url	PG	The URL to a map of the producer.
ProductAttribute_Id	PAID	The unique id for the attribute of the product.
ProductAttribute_Name	PAN	The name, or description, of the attribute. Ex: Boutique Wines
ProductAttribute_Url	PAU	The URL to other products that have the same attribute. Can be null.
ProductAttribute_ImageUrl	PAIU	The URL to the image representing this attribute. Can be null.

Table A.2: Wine.com properties

Appendix B

Wine Schema Dependencies

B.1 Functional dependencies

$$wineProduct : \{WPID\} \rightarrow \{YWID, WPU, WPM, WPX, WPR, WPT, WPV, BT\}$$

$$wineProductWithGeoLocation : \{WPID\} \rightarrow \{WPG\} \quad \text{(B.2)}$$
$$wineProduct : \{YWID, BT\} \rightarrow \{WPID\} \quad \text{(B.3)}$$
$$appellation : \{AID\} \rightarrow \{AN, RID\} \quad \text{(B.4)}$$
$$region : \{RID\} \rightarrow \{RN\} \quad \text{(B.5)}$$
$$varietalBlend : \{VBID\} \rightarrow \{VBN, TID\} \quad \text{(B.6)}$$
$$wineType : \{TID\} \rightarrow \{TN\} \quad \text{(B.7)}$$
$$producer : \{PID\} \rightarrow \{PN, PU\} \quad \text{(B.8)}$$
$$producerWithImage : \{PID\} \rightarrow \{PIU\} \quad \text{(B.9)}$$
$$producerWithGeoLocation : \{PID\} \rightarrow \{PG\} \quad \text{(B.10)}$$
$$productAttribute : \{PAID\} \rightarrow \{PAN\} \quad \text{(B.11)}$$
$$productAttributeWithImage : \{PAID\} \rightarrow \{PAIU\} \quad \text{(B.12)}$$
$$wine : \{WID\} \rightarrow \{WPN, PID\} \quad \text{(B.13)}$$
$$wine : \{WPN\} \rightarrow \{WID\} \quad \text{(B.14)}$$
$$yearlyWine : \{YWID\} \rightarrow \{WID, WPY, AID, VBID\} \quad \text{(B.15)}$$
$$yearlyWine : \{WPN, WPY\} \rightarrow \{YWID\} \quad \text{(B.16)}$$

B.2 Multivalued dependencies

$$hasAttribute : \{WPID\} \twoheadrightarrow \{PAID\} \quad \text{(B.17)}$$

B.3 Inclusion dependencies

$$appellation[RID] \subseteq region[RID] \tag{B.18}$$
$$hasAttribute[PAID] \subseteq productAttribute[PAID] \tag{B.19}$$
$$hasAttribute[WPID] \subseteq wineProduct[WPID] \tag{B.20}$$
$$producerWithGeoLocation[PID] \subseteq Producer[PID] \tag{B.21}$$
$$producerWithImage[PID] \subseteq producer[PID] \tag{B.22}$$
$$productAttributeWithImage[PAID] \subseteq productAttribute[PAID] \tag{B.23}$$
$$varietalBlend[TID] \subseteq wineType[TID] \tag{B.24}$$
$$wine[PID] \subseteq producer[PID] \tag{B.25}$$
$$wineProduct[YWID] \subseteq yearlyWine[YWID] \tag{B.26}$$
$$wineProductWithGeoLocation[WPID] \subseteq wineProduct[WPID] \tag{B.27}$$
$$yearlyWine[AID] \subseteq appellation[AID] \tag{B.28}$$
$$yearlyWine[VBID] \subseteq varietalBlend[VBID] \tag{B.29}$$
$$yearlyWine[WID] \subseteq wine[WID] \tag{B.30}$$

Appendix C

EBNF Grammars for ORM*

C.1 ORM2plus grammar

⟨*Schema*⟩ ::= ⟨*Entities*⟩ ⟨*Values*⟩ ⟨*Relations*⟩ ⟨*Constraint*⟩*

⟨*Entities*⟩ ::= 'ENTITYTYPES:{' ⟨*EntityName*⟩(',' ⟨*EntityName*⟩)* '}'

⟨*Values*⟩ ::= 'VALUETYPES:{' ⟨*ValueName*⟩(',' ⟨*ValueName*⟩)* '}'

⟨*Relations*⟩ ::= 'RELATIONS:{' ⟨*RelationName*⟩(',' ⟨*RelationName*⟩)* '}'

⟨*EntityName*⟩ ::= ⟨*Identifier*⟩

⟨*ValueName*⟩ ::= ⟨*Identifier*⟩

⟨*RelationName*⟩ ::= ⟨*Identifier*⟩

⟨*Identifier*⟩ ::= [a-zA-Z][a-zA-Z_]*

⟨*Constraint*⟩ ::= ⟨*Type*⟩ | ⟨*Freq*⟩ | ⟨*Mand*⟩ | ⟨*RSet*⟩ | ⟨*OSet*⟩ | ⟨*OCard*⟩ | ⟨*RCard*⟩ | ⟨*Obj*⟩ | ⟨*Ring*⟩ | ⟨*VVal*⟩

⟨*Type*⟩ ::= 'TYPE(' ⟨*LocRole*⟩ ',' ⟨*ObjectName*⟩ ')'

⟨*LocRole*⟩ ::= ⟨*RelationName*⟩ '.' (⟨*Int*⟩ | ⟨*ObjectName*⟩)

⟨*ObjectName*⟩ :: = ⟨*EntityName*⟩ | ⟨*ValueName*⟩

⟨*Freq*⟩ ::= 'FREQ(' ⟨*LocRoleSet*⟩ ',' ⟨*Joins*⟩ ',<' ⟨*MinVal*⟩ ',' ⟨*MaxValInf*⟩ '>)'

⟨*LocRoleSet*⟩ ::= '{' ⟨*LocRole*⟩(',' ⟨*LocRole*⟩)* '}'

⟨*Joins*⟩ ::= '{' ⟨*LocRoleJoin*⟩(',' ⟨*LocRoleJoin*⟩)* '}'

⟨*LocRoleJoin*⟩ ::= '<' ⟨*LocRole*⟩ '=' ⟨*LocRole*⟩ '>'

⟨*MinVal*⟩ ::= ⟨*Int*⟩

⟨*MaxValInf*⟩ ::= ⟨*Int*⟩ | 'inf'

⟨*Mand*⟩ ::= 'MAND(' ⟨*LocRoleSet*⟩ ',' ⟨*ObjectName*⟩ ')'

⟨*RSet*⟩ ::= 'R-SET' ⟨*RSetType*⟩ '((' ⟨*LocRoleSet*⟩ ',' ⟨*Joins*⟩ '),(' ⟨*LocRoleSet*⟩ ',' ⟨*Joins*⟩ '),{' ⟨*LocRoleTuple*⟩(',' ⟨*LocRoleTuple*⟩)* '})'

⟨*RSetType*⟩ ::= 'Exc' | 'Sub'

⟨*LocRoleTuple*⟩ ::= '(' ⟨*LocRole*⟩ ',' ⟨*LocRole*⟩ ')'

⟨*OSet*⟩ ::= 'O-SET' ⟨*OSetType*⟩ '({' ⟨*ObjectName*⟩(',' ⟨*ObjectName*⟩)* '},' ⟨*ObjectName*⟩ ')'

⟨*OSetType*⟩ ::= 'Isa' | 'Ex' | 'Tot'

⟨*OCard*⟩ ::= 'O-CARD(' ⟨*ObjectName*⟩ ') = (' ⟨*MinVal*⟩ ',' ⟨*MaxValInf*⟩ ')'

⟨*RCard*⟩ ::= 'R-CARD(' ⟨*LocRole*⟩ ') = (' ⟨*MinVal*⟩ ',' ⟨*MaxValInf*⟩ ')'

⟨*Obj*⟩ ::= 'OBJ(' ⟨*RelationName*⟩ ',' ⟨*ObjectName*⟩ ')'

⟨*Ring*⟩ ::= 'RING' ⟨*RingType*⟩ '(' ⟨*LocRole*⟩ ',' ⟨*LocRole*⟩ ')'

⟨*RingType*⟩ ::= 'Irr' | 'Asym' | 'Trans' | 'Intr' | 'Antisym' | 'Acyclic' | 'Sym' | 'Ref' | ...

⟨*VVal*⟩ ::= 'V-VAL(' ⟨*ValueName*⟩ ') = {' (⟨*Int*⟩(',' ⟨*Int*⟩)* | ⟨*QuotedString*⟩(',' ⟨*QuotedString*⟩)*) '}'

⟨*Int*⟩ ::= [1-9][0-9]*

⟨*QuotedString*⟩ ::= ' ‘' [^’]+ '’ '

C.2 ORM2^{zero} grammar

Two production rules are defined for the constructs FREQ^{-} and R-SET^{-}. Also ⟨*Constraint*⟩ is redefined.

⟨*FreqMinus*⟩ ::= 'FREQ^{-}(' ⟨*LocRole*⟩ ',<' ⟨*MaxVal*⟩ ',' ⟨*MaxValInf*⟩ '>)'

⟨*RSetMinus*⟩ ::= 'R-SET^{-}' ⟨*RSetType*⟩ '(' ⟨*LocRoleSet*⟩ ',' ⟨*LocRoleSet*⟩ ')'

⟨*Constraint*⟩ ::= ⟨*Type*⟩ | ⟨*FreqMinus*⟩ | ⟨*Mand*⟩ | ⟨*RSetMinus*⟩ | ⟨*OSet*⟩ | ⟨*Obj*⟩

C.3 ORM2^{bin} grammar

The main difference between ORM2^{zero} and ORM2^{bin} is the redefinition of ⟨*Constraint*⟩ to remove ⟨*Obj*⟩ and add ⟨*Primary*⟩ and ⟨*Label*⟩ .

⟨*Constraint*⟩ ::= ⟨*Type*⟩ | ⟨*FreqMinus*⟩ | ⟨*Mand*⟩ | ⟨*RSetMinus*⟩ | ⟨*OSet*⟩ | ⟨*Primary*⟩
| ⟨*Label*⟩

⟨*Primary*⟩ ::= `'PRIMARY('` ⟨*ObjectName*⟩ `','` ⟨*LocRole*⟩ `')'`

⟨*Label*⟩ ::= `'LABEL('` ⟨*ObjectName*⟩ `','` ⟨*LocRole*⟩ `')'`

Appendix D

Wine Schema in ORM2^{bin} Syntax

```
1  RELATIONS: {hasProducerUrl,hasMinPrice,hasDetailUrl
       ,hasProducerName,hasAttributeName,hasRetailPrice
       ,hasVarietalOrBlendName,hasMaxPrice,
       isOfVarietalOrBlend,hasWineTypeName,hasImageUrl,
       belongsToAppellation,isOfWine,
       hasHighestRatingScore,isOfVintageYear,
       hasRegionName,hasAppellationName,hasMapUrl,
       hasProducerImageUrl,createdByProducer,
       hasAttribute,correspondsToTypeOfWine,hasWineName
       ,hasHighestReviewScore,isOfBottleType,
       isOfYearlyWine,hasLocationUrl,
       correspondsToRegion}
2  ENTITYTYPES: {Producer,WineProductWithGeoLocation,
       VarietalBlend,ProductAttribute,WineProduct,
       ProductAttributeWithImage,Appellation,YearlyWine
       ,Wine,WineType,ProducerWithImage,Region,
       ProducerWithGeoLocation}
3  VALUETYPES: {Url,VarietalBlendName,Year,ImageUrl,
       TypeName,Price,ReviewsScore,RegionName,
       ProducerName,AttributeName,AppellationName,
       RatingsScore,BottleType,WineName,ImageUrl}
4
5  FREQ-(belongsToAppellation.1, <1,1>)
6  FREQ-(correspondsToRegion.1, <1,1>)
7  FREQ-(correspondsToTypeOfWine.1, <1,1>)
8  FREQ-(createdByProducer.1, <1,1>)
9  FREQ-(hasAppellationName.1, <1,1>)
10 FREQ-(hasAttributeName.1, <1,1>)
11 FREQ-(hasDetailUrl.1, <1,1>)
12 FREQ-(hasHighestRatingScore.1, <1,1>)
13 FREQ-(hasHighestReviewScore.1, <1,1>)
14 FREQ-(hasImageUrl.1, <1,1>)
15 FREQ-(hasLocationUrl.1, <1,1>)
16 FREQ-(hasMapUrl.1, <1,1>)
17 FREQ-(hasMaxPrice.1, <1,1>)
18 FREQ-(hasMinPrice.1, <1,1>)
19 FREQ-(hasProducerImageUrl.1, <1,1>)
```

```
20 FREQ(hasProducerName.1, <1,1>)
21 FREQ(hasProducerUrl.1, <1,1>)
22 FREQ(hasRegionName.1, <1,1>)
23 FREQ(hasRetailPrice.1, <1,1>)
24 FREQ(hasVarietalOrBlendName.1, <1,1>)
25 FREQ(hasWineName.1, <1,1>)
26 FREQ(hasWineTypeName.1, <1,1>)
27 FREQ(isOfBottleType.1, <1,1>)
28 FREQ(isOfVarietalOrBlend.1, <1,1>)
29 FREQ(isOfVintageYear.1, <1,1>)
30 FREQ(isOfWine.1, <1,1>)
31 FREQ(isOfYearlyWine.1, <1,1>)
32
33 MAND({belongsToAppellation.1}, YearlyWine)
34 MAND({correspondsToRegion.1}, Appellation)
35 MAND({correspondsToTypeOfWine.1}, VarietalBlend)
36 MAND({createdByProducer.1}, Wine)
37 MAND({hasAppellationName.1}, Appellation)
38 MAND({hasAttributeName.1}, ProductAttribute)
39 MAND({hasDetailUrl.1}, WineProduct)
40 MAND({hasHighestRatingScore.1}, WineProduct)
41 MAND({hasHighestReviewScore.1}, WineProduct)
42 MAND({hasImageUrl.1}, ProductAttributeWithImage)
43 MAND({hasLocationUrl.1}, WineProductWithGeoLocation
       )
44 MAND({hasMapUrl.1}, ProducerWithGeoLocation)
45 MAND({hasMaxPrice.1}, WineProduct)
46 MAND({hasMinPrice.1}, WineProduct)
47 MAND({hasProducerImageUrl.1}, ProducerWithImage)
48 MAND({hasProducerName.1}, Producer)
49 MAND({hasProducerUrl.1}, Producer)
50 MAND({hasRegionName.1}, Region)
51 MAND({hasRetailPrice.1}, WineProduct)
52 MAND({hasVarietalOrBlendName.1}, VarietalBlend)
53 MAND({hasWineName.1}, Wine)
54 MAND({hasWineTypeName.1}, WineType)
55 MAND({isOfBottleType.1}, WineProduct)
56 MAND({isOfVarietalOrBlend.1}, YearlyWine)
57 MAND({isOfVintageYear.1}, YearlyWine)
58 MAND({isOfWine.1}, YearlyWine)
59 MAND({isOfYearlyWine.1}, WineProduct)
60
61 TYPE(belongsToAppellation.1, YearlyWine)
62 TYPE(belongsToAppellation.2, Appellation)
63 TYPE(correspondsToRegion.1, Appellation)
64 TYPE(correspondsToRegion.2, Region)
65 TYPE(correspondsToTypeOfWine.1, VarietalBlend)
66 TYPE(correspondsToTypeOfWine.2, WineType)
67 TYPE(createdByProducer.1, Wine)
68 TYPE(createdByProducer.2, Producer)
69 TYPE(hasAppellationName.1, Appellation)
70 TYPE(hasAppellationName.2, AppellationName)
71 TYPE(hasAttribute.1, WineProduct)
72 TYPE(hasAttribute.2, ProductAttribute)
73 TYPE(hasAttributeName.1, ProductAttribute)
74 TYPE(hasAttributeName.2, AttributeName)
```

```
75  TYPE(hasDetailUrl.1, WineProduct)
76  TYPE(hasDetailUrl.2, Url)
77  TYPE(hasHighestRatingScore.1, WineProduct)
78  TYPE(hasHighestRatingScore.2, RatingsScore)
79  TYPE(hasHighestReviewScore.1, WineProduct)
80  TYPE(hasHighestReviewScore.2, ReviewsScore)
81  TYPE(hasImageUrl.1, ProductAttributeWithImage)
82  TYPE(hasImageUrl.2, ImageUrl)
83  TYPE(hasLocationUrl.1, WineProductWithGeoLocation)
84  TYPE(hasLocationUrl.2, Url)
85  TYPE(hasMapUrl.1, ProducerWithGeoLocation)
86  TYPE(hasMapUrl.2, Url)
87  TYPE(hasMaxPrice.1, WineProduct)
88  TYPE(hasMaxPrice.2, Price)
89  TYPE(hasMinPrice.1, WineProduct)
90  TYPE(hasMinPrice.2, Price)
91  TYPE(hasProducerImageUrl.1, ProducerWithImage)
92  TYPE(hasProducerImageUrl.2, ImageUrl)
93  TYPE(hasProducerName.1, Producer)
94  TYPE(hasProducerName.2, ProducerName)
95  TYPE(hasProducerUrl.1, Producer)
96  TYPE(hasProducerUrl.2, Url)
97  TYPE(hasRegionName.1, Region)
98  TYPE(hasRegionName.2, RegionName)
99  TYPE(hasRetailPrice.1, WineProduct)
100 TYPE(hasRetailPrice.2, Price)
101 TYPE(hasVarietalOrBlendName.1, VarietalBlend)
102 TYPE(hasVarietalOrBlendName.2, VarietalBlendName)
103 TYPE(hasWineName.1, Wine)
104 TYPE(hasWineName.2, WineName)
105 TYPE(hasWineTypeName.1, WineType)
106 TYPE(hasWineTypeName.2, TypeName)
107 TYPE(isOfBottleType.1, WineProduct)
108 TYPE(isOfBottleType.2, BottleType)
109 TYPE(isOfVarietalOrBlend.1, YearlyWine)
110 TYPE(isOfVarietalOrBlend.2, VarietalBlend)
111 TYPE(isOfVintageYear.1, YearlyWine)
112 TYPE(isOfVintageYear.2, Year)
113 TYPE(isOfWine.1, YearlyWine)
114 TYPE(isOfWine.2, Wine)
115 TYPE(isOfYearlyWine.1, WineProduct)
116 TYPE(isOfYearlyWine.2, YearlyWine)
117
118 O-SET_Isa({ProducerWithGeoLocation,
        ProducerWithImage}, Producer)
119 O-SET_Isa({ProductAttributeWithImage},
        ProductAttribute)
120 O-SET_Isa({WineProductWithGeoLocation}, WineProduct
        )
121
122 % Not allowed in ORM Zero
123 % FREQ({isOfBottleType.2, isOfYearlyWine.2}, {<
        isOfBottleType.1=isOfYearlyWine.2>}, <1, 1>)
124 % FREQ({isOfVintageYear.2, isOfWine.2}, {<
        isOfVintageYear.1=isOfWine.1>}, <1, 1>)
125
```

```
126 % V-VAL(Price)={0,..,inf}
127 % V-VAL(RatingsScore)={0,..,100}
128 % V-VAL(ReviewsScore)={0,..,5}
129 % V-VAL(TypeName)={ Red Wine, Rose Wine, Champagne
      \&amp; Sparkling, Dessert, Sherry \&amp; Port,
      White Wine, Sake,}
130
131 % Frequency constraints in ORM Plus syntax
132 % FREQ({belongsToAppellation.1}, {}, <1, 1>)
133 % FREQ({correspondsToRegion.1}, {}, <1, 1>)
134 % FREQ({correspondsToTypeOfWine.1}, {}, <1, 1>)
135 % FREQ({createdByProducer.1}, {}, <1, 1>)
136 % FREQ({hasAppellationName.1}, {}, <1, 1>)
137 % FREQ({hasAttributeName.1}, {}, <1, 1>)
138 % FREQ({hasDetailUrl.1}, {}, <1, 1>)
139 % FREQ({hasHighestRatingScore.1}, {}, <1, 1>)
140 % FREQ({hasHighestReviewScore.1}, {}, <1, 1>)
141 % FREQ({hasImageUrl.1}, {}, <1, 1>)
142 % FREQ({hasLocationUrl.1}, {}, <1, 1>)
143 % FREQ({hasMapUrl.1}, {}, <1, 1>)
144 % FREQ({hasMaxPrice.1}, {}, <1, 1>)
145 % FREQ({hasMinPrice.1}, {}, <1, 1>)
146 % FREQ({hasProducerImageUrl.1}, {}, <1, 1>)
147 % FREQ({hasProducerName.1}, {}, <1, 1>)
148 % FREQ({hasProducerUrl.1}, {}, <1, 1>)
149 % FREQ({hasRegionName.1}, {}, <1, 1>)
150 % FREQ({hasRetailPrice.1}, {}, <1, 1>)
151 % FREQ({hasVarietalOrBlendName.1}, {}, <1, 1>)
152 % FREQ({hasWineName.1}, {}, <1, 1>)
153 % FREQ({hasWineTypeName.1}, {}, <1, 1>)
154 % FREQ({isOfBottleType.1}, {}, <1, 1>)
155 % FREQ({isOfVarietalOrBlend.1}, {}, <1, 1>)
156 % FREQ({isOfVintageYear.1}, {}, <1, 1>)
157 % FREQ({isOfWine.1}, {}, <1, 1>)
158 % FREQ({isOfYearlyWine.1}, {}, <1, 1>)
```

Appendix E

Wine.com Ontology

```
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(:=<http://www.semanticweb.org/xikjc/2015/3/
    WineCom.owl#>)
Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-
    syntax-ns#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema
    #>)


Ontology(<http://www.semanticweb.org/xikjc/2015/3/
    WineCom.owl>

Declaration(Class(:Appellation))
SubClassOf(:Appellation ObjectMinCardinality(1 :
    correspondsToRegion))
Declaration(Class(:Producer))
SubClassOf(:Producer DataMinCardinality(1 :
    hasProducerUrl))
Declaration(Class(:ProducerWithGeoLocation))
SubClassOf(:ProducerWithGeoLocation :Producer)
SubClassOf(:ProducerWithGeoLocation
    DataMinCardinality(1 :hasMapUrl))
Declaration(Class(:ProducerWithImage))
SubClassOf(:ProducerWithImage :Producer)
SubClassOf(:ProducerWithImage DataMinCardinality(1
    :hasProducerImageUrl))
Declaration(Class(:ProductAttribute))
Declaration(Class(:ProductAttributeWithImage))
SubClassOf(:ProductAttributeWithImage :
    ProductAttribute)
SubClassOf(:ProductAttributeWithImage
    DataMinCardinality(1 :hasImageUrl))
Declaration(Class(:Region))
Declaration(Class(:VarietalBlend))
SubClassOf(:VarietalBlend ObjectMinCardinality(1 :
    correspondsToTypeOfWine))
Declaration(Class(:Wine))
```

```
SubClassOf(:Wine ObjectMinCardinality(1 :
    createdByProducer))
Declaration(Class(:WineProduct))
SubClassOf(:WineProduct ObjectMinCardinality(1 :
    isOfYearlyWine))
SubClassOf(:WineProduct DataMinCardinality(1 :
    hasDetailUrl))
SubClassOf(:WineProduct DataMinCardinality(1 :
    hasHighestRatingScore))
SubClassOf(:WineProduct DataMinCardinality(1 :
    hasHighestReviewScore))
SubClassOf(:WineProduct DataMinCardinality(1 :
    hasMaxPrice))
SubClassOf(:WineProduct DataMinCardinality(1 :
    hasMinPrice))
SubClassOf(:WineProduct DataMinCardinality(1 :
    hasRetailPrice))
SubClassOf(:WineProduct DataMinCardinality(1 :
    isOfBottleType))
Declaration(Class(:WineProductWithGeoLocation))
SubClassOf(:WineProductWithGeoLocation :WineProduct
    )
SubClassOf(:WineProductWithGeoLocation
    DataMinCardinality(1 :hasLocationUrl))
Declaration(Class(:WineType))
Declaration(Class(:YearlyWine))
SubClassOf(:YearlyWine ObjectMinCardinality(1 :
    belongsToAppellation))
SubClassOf(:YearlyWine ObjectMinCardinality(1 :
    isOfVarietalOrBlend))
SubClassOf(:YearlyWine ObjectMinCardinality(1 :
    isOfWine))
SubClassOf(:YearlyWine DataMinCardinality(1 :
    isOfVintageYear))
Declaration(ObjectProperty(:belongsToAppellation))
ObjectPropertyDomain(:belongsToAppellation :
    YearlyWine)
ObjectPropertyDomain(:belongsToAppellation
    ObjectMinCardinality(0 :belongsToAppellation))
ObjectPropertyDomain(:belongsToAppellation
    ObjectMaxCardinality(1 :belongsToAppellation))
ObjectPropertyRange(:belongsToAppellation :
    Appellation)
Declaration(ObjectProperty(:correspondsToRegion))
ObjectPropertyDomain(:correspondsToRegion :
    Appellation)
ObjectPropertyDomain(:correspondsToRegion
    ObjectMinCardinality(0 :correspondsToRegion))
ObjectPropertyDomain(:correspondsToRegion
    ObjectMaxCardinality(1 :correspondsToRegion))
ObjectPropertyRange(:correspondsToRegion :Region)
Declaration(ObjectProperty(:correspondsToTypeOfWine
    ))
ObjectPropertyDomain(:correspondsToTypeOfWine :
    VarietalBlend)
ObjectPropertyDomain(:correspondsToTypeOfWine
    ObjectMinCardinality(0 :correspondsToTypeOfWine)
    )
```

```
ObjectPropertyDomain(:correspondsToTypeOfWine
    ObjectMaxCardinality(1 :correspondsToTypeOfWine)
    )
ObjectPropertyRange(:correspondsToTypeOfWine :
    WineType)
Declaration(ObjectProperty(:createdByProducer))
ObjectPropertyDomain(:createdByProducer :Wine)
ObjectPropertyDomain(:createdByProducer
    ObjectMinCardinality(0 :createdByProducer))
ObjectPropertyDomain(:createdByProducer
    ObjectMaxCardinality(1 :createdByProducer))
ObjectPropertyRange(:createdByProducer :Producer)
Declaration(ObjectProperty(:hasAttribute))
ObjectPropertyDomain(:hasAttribute :WineProduct)
ObjectPropertyRange(:hasAttribute :ProductAttribute
    )
Declaration(ObjectProperty(:isOfVarietalOrBlend))
ObjectPropertyDomain(:isOfVarietalOrBlend :
    YearlyWine)
ObjectPropertyDomain(:isOfVarietalOrBlend
    ObjectMinCardinality(0 :isOfVarietalOrBlend))
ObjectPropertyDomain(:isOfVarietalOrBlend
    ObjectMaxCardinality(1 :isOfVarietalOrBlend))
ObjectPropertyRange(:isOfVarietalOrBlend :
    VarietalBlend)
Declaration(ObjectProperty(:isOfWine))
ObjectPropertyDomain(:isOfWine :YearlyWine)
ObjectPropertyDomain(:isOfWine ObjectMinCardinality
    (0 :isOfWine))
ObjectPropertyDomain(:isOfWine ObjectMaxCardinality
    (1 :isOfWine))
ObjectPropertyRange(:isOfWine :Wine)
Declaration(ObjectProperty(:isOfYearlyWine))
ObjectPropertyDomain(:isOfYearlyWine :WineProduct)
ObjectPropertyDomain(:isOfYearlyWine
    ObjectMinCardinality(0 :isOfYearlyWine))
ObjectPropertyDomain(:isOfYearlyWine
    ObjectMaxCardinality(1 :isOfYearlyWine))
ObjectPropertyRange(:isOfYearlyWine :YearlyWine)
Declaration(DataProperty(:hasDetailUrl))
DataPropertyDomain(:hasDetailUrl :WineProduct)
DataPropertyDomain(:hasDetailUrl DataMinCardinality
    (0 :hasDetailUrl))
DataPropertyDomain(:hasDetailUrl DataMaxCardinality
    (1 :hasDetailUrl))
DataPropertyRange(:hasDetailUrl :Url)
Declaration(DataProperty(:hasHighestRatingScore))
DataPropertyDomain(:hasHighestRatingScore :
    WineProduct)
DataPropertyDomain(:hasHighestRatingScore
    DataMinCardinality(0 :hasHighestRatingScore))
DataPropertyDomain(:hasHighestRatingScore
    DataMaxCardinality(1 :hasHighestRatingScore))
DataPropertyRange(:hasHighestRatingScore :
    RatingsScore)
Declaration(DataProperty(:hasHighestReviewScore))
```

```
DataPropertyDomain(:hasHighestReviewScore :
    WineProduct)
DataPropertyDomain(:hasHighestReviewScore
    DataMinCardinality(0 :hasHighestReviewScore))
DataPropertyDomain(:hasHighestReviewScore
    DataMaxCardinality(1 :hasHighestReviewScore))
DataPropertyRange(:hasHighestReviewScore :
    ReviewsScore)
Declaration(DataProperty(:hasImageUrl))
DataPropertyDomain(:hasImageUrl :
    ProductAttributeWithImage)
DataPropertyDomain(:hasImageUrl DataMinCardinality
    (0 :hasImageUrl))
DataPropertyDomain(:hasImageUrl DataMaxCardinality
    (1 :hasImageUrl))
DataPropertyRange(:hasImageUrl :ImageUrl)
Declaration(DataProperty(:hasLocationUrl))
DataPropertyDomain(:hasLocationUrl :
    WineProductWithGeoLocation)
DataPropertyDomain(:hasLocationUrl
    DataMinCardinality(0 :hasLocationUrl))
DataPropertyDomain(:hasLocationUrl
    DataMaxCardinality(1 :hasLocationUrl))
DataPropertyRange(:hasLocationUrl :Url)
Declaration(DataProperty(:hasMapUrl))
DataPropertyDomain(:hasMapUrl :
    ProducerWithGeoLocation)
DataPropertyDomain(:hasMapUrl DataMinCardinality(0
    :hasMapUrl))
DataPropertyDomain(:hasMapUrl DataMaxCardinality(1
    :hasMapUrl))
DataPropertyRange(:hasMapUrl :Url)
Declaration(DataProperty(:hasMaxPrice))
DataPropertyDomain(:hasMaxPrice :WineProduct)
DataPropertyDomain(:hasMaxPrice DataMinCardinality
    (0 :hasMaxPrice))
DataPropertyDomain(:hasMaxPrice DataMaxCardinality
    (1 :hasMaxPrice))
DataPropertyRange(:hasMaxPrice :Price)
Declaration(DataProperty(:hasMinPrice))
DataPropertyDomain(:hasMinPrice :WineProduct)
DataPropertyDomain(:hasMinPrice DataMinCardinality
    (0 :hasMinPrice))
DataPropertyDomain(:hasMinPrice DataMaxCardinality
    (1 :hasMinPrice))
DataPropertyRange(:hasMinPrice :Price)
Declaration(DataProperty(:hasProducerImageUrl))
DataPropertyDomain(:hasProducerImageUrl :
    ProducerWithImage)
DataPropertyDomain(:hasProducerImageUrl
    DataMinCardinality(0 :hasProducerImageUrl))
DataPropertyDomain(:hasProducerImageUrl
    DataMaxCardinality(1 :hasProducerImageUrl))
DataPropertyRange(:hasProducerImageUrl :ImageUrl)
Declaration(DataProperty(:hasProducerUrl))
DataPropertyDomain(:hasProducerUrl :Producer)
DataPropertyDomain(:hasProducerUrl
    DataMinCardinality(0 :hasProducerUrl))
```

```
DataPropertyDomain(:hasProducerUrl
    DataMaxCardinality(1 :hasProducerUrl))
DataPropertyRange(:hasProducerUrl :Url)
Declaration(DataProperty(:hasRetailPrice))
DataPropertyDomain(:hasRetailPrice :WineProduct)
DataPropertyDomain(:hasRetailPrice
    DataMinCardinality(0 :hasRetailPrice))
DataPropertyDomain(:hasRetailPrice
    DataMaxCardinality(1 :hasRetailPrice))
DataPropertyRange(:hasRetailPrice :Price)
Declaration(DataProperty(:isOfBottleType))
DataPropertyDomain(:isOfBottleType :WineProduct)
DataPropertyDomain(:isOfBottleType
    DataMinCardinality(0 :isOfBottleType))
DataPropertyDomain(:isOfBottleType
    DataMaxCardinality(1 :isOfBottleType))
DataPropertyRange(:isOfBottleType :BottleType)
Declaration(DataProperty(:isOfVintageYear))
DataPropertyDomain(:isOfVintageYear :YearlyWine)
DataPropertyDomain(:isOfVintageYear
    DataMinCardinality(0 :isOfVintageYear))
DataPropertyDomain(:isOfVintageYear
    DataMaxCardinality(1 :isOfVintageYear))
DataPropertyRange(:isOfVintageYear :Year)
Declaration(Datatype(:BottleType))
DatatypeDefinition(:BottleType xsd:string)
Declaration(Datatype(:ImageUrl))
DatatypeDefinition(:ImageUrl xsd:string)
Declaration(Datatype(:Price))
DatatypeDefinition(:Price xsd:decimal)
Declaration(Datatype(:RatingsScore))
DatatypeDefinition(:RatingsScore xsd:integer)
Declaration(Datatype(:ReviewsScore))
DatatypeDefinition(:ReviewsScore xsd:integer)
Declaration(Datatype(:Url))
DatatypeDefinition(:Url xsd:string)
Declaration(Datatype(:Year))
DatatypeDefinition(:Year xsd:integer)
DisjointClasses(:Appellation :Producer :
    ProductAttribute :Region :VarietalBlend :Wine :
    WineProduct :WineType :YearlyWine)
)
```

Bibliography

[1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003. ISBN 0-521-78176-0.

[2] P. A. Bernstein. Synthesizing third normal form relations from functional dependencies. *ACM Transactions on Database Systems*, 1(4):277–298, Dec. 1976. ISSN 03625915. doi: 10.1145/320493.320489. URL `http://portal.acm.org/citation.cfm?doid=320493.320489`.

[3] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, June 1970. doi: 10.1145/362384.362685.

[4] S. David, A. Ercolani, E. Franconi, L. Lubyte, M. Faheem, and S. Tessaris. A Methodology to Devise Conceptual Schemas from Raw Data Schema Analysis. 2010.

[5] O. De Troyer. Formalization of the binary object-role model based on logic. *Data and Knowledge Engineering*, 19:1–37, 1996. ISSN 0169023X. doi: 10.1016/0169-023X(95)00045-T.

[6] J. Demey, M. Jarrar, and R. Meersman. A Conceptual Markup Language That Supports Interoperability between Business Rule Modeling Systems. In *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE : Confederated International Conferences CoopIS, DOA, and ODBASE 2002. Proceedings*, pages 19–35, 2002. ISBN 3-540-00106-9. URL `http://www.springerlink.com/content/dc3fdjcwtcwj6dgu`.

[7] R. Fagin. Multivalued Dependencies and a New Form for Relational Databases. *ACM Transactions on Database System*, 2(3):262–278, 1977. doi: 10.1145/320557.320571.

[8] E. Franconi and A. Mosca. The formalisation of ORM2 and its encoding in OWL2. Technical report, KRDB Research Centre, Free University of Bozen-Bolzano, 2012. URL `http://www.inf.unibz.it/krdb/pub/TR/KRDB12-2.pdf`.

[9] E. Franconi, P. Guagliardo, S. Tessaris, and M. Trevisan. A natural language ontology-driven query interface. In *9th International Conference on Terminology and Artificial Intelligence*, page 43, 2011.

[10] E. Franconi, A. Mosca, and D. Solomakhin. ORM2: Formalisation and Encoding in OWL2. In P. Herrero, H. Panetto, R. Meersman, and T. Dillon, editors, *On the Move to Meaningful Internet Systems: OTM 2012 Workshops*, volume 7567 of *Lecture Notes in Computer Science*, pages 368–378. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33617-1. doi: 10.1007/978-3-642-33618-8_51. URL `http://dx.doi.org/10.1007/978-3-642-33618-8_51`.

[11] T. Halpin. *A Logical Analysis of Information Systems: Static Aspects of the Data-oriented Perspective*. Phd thesis, University of Queensland, 1989.

[12] T. Halpin. Object-Role Modeling. In P. Bernus, K. Mertins, and G. Schmidt, editors, *Handbook on Architectures of Information Systems*, chapter 4. Springer-Verlag, 1998. ISBN 3540644539.

[13] T. Halpin. ORM 2. In R. Meersman, Z. Tari, and P. Herrero, editors, *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, volume 3762 of *Lecture Notes in Computer Science*, pages 676–687. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-29739-0. doi: 10.1007/11575863_87. URL `http://dx.doi.org/10.1007/11575863_87`.

[14] T. Halpin. Ontological Modeling : Part 4. *Business Rules Journal*, 11(6): 1–7, June 2010. URL `http://www.brcommunity.com/a2010/b539.html`.

[15] T. Halpin. Ontological Modeling : Part 5. *Business Rules Journal*, 11(12): 1–8, Dec. 2010. URL `http://www.brcommunity.com/a2010/b570.html`.

[16] T. Halpin. Ontological Modeling : Part 6. *Business Rules Journal*, 12(2): 1–7, Feb. 2011. URL `http://www.brcommunity.com/a2011/b579.html`.

[17] T. Halpin. Ontological Modeling : Part 13. *Business Rules Journal*, 14(3): 1–9, Mar. 2013. URL `http://www.brcommunity.com/a2013/b693.html`.

[18] T. A. Halpin and H. A. Proper. Subtyping and Polymorphism in Object-Role Modelling. *Data & Knowledge Engineering*, 15(3):251–281, 1995.

[19] P. Hayes and P. F. Patel-Schneider. RDF 1.1 Semantics. W3C recommendation, W3C, Feb. 2014. URL `http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/`.

[20] R. Hodrob and M. Jarrar. Mapping ORM into OWL 2. In *Proceedings of the 1st International Conference on Intelligent Semantic Web-Services and Applications - ISWSA '10*, pages 1–7, New York, New York, USA, 2010. ACM Press. ISBN 9781450304757. doi: 10.1145/1874590.1874599. URL `http://portal.acm.org/citation.cfm?doid=1874590.1874599`.

[21] M. Jarrar. Towards Automated Reasoning on ORM Schemes Mapping ORM into the DLR idf Description Logic. In *International Conference on Conceptual Modeling*, pages 181–197. Springer, 2007. ISBN 3-540-75562-4, 978-3-540-75562-3.

[22] M. Jarrar. Mapping ORM into the SHOIN/OWL Description Logic. pages 729–741. Springer, 2007.

[23] X. I. K. Juarez-Castro. Presenting query results in NL for Quelo. Technical report, Faculty of Computer Science, Free University of Bozen-Bolzano, 2014.

[24] C. M. Keet. Mapping the Object-Role Modeling language ORM2 into Description Logic language DLRifd. *CoRR*, abs/cs/070, 2007. URL `http://arxiv.org/abs/cs/0702089`.

[25] M. Krötzsch, F. Simančík, and I. Horrocks. A Description Logic Primer. *CoRR*, abs/1201.4(June):1–17, 2012. URL `http://arxiv.org/abs/1201.4089`.

[26] C. Løvdahl. *A Comparison of Information Modeling in ORM and OWL*. Master thesis, 2011.

[27] D. Maier. Null Values, Partial Information and Database Semantics. In *The Theory of Relational Databases*, chapter 12, pages 371–438. Computer Science Press, 1983. ISBN 0914894420.

[28] B. Motik, B. Parsia, and P. F. Patel-Schneider. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition). W3C recommendation, W3C, Dec. 2012. URL `http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/`.

[29] B. Motik, P. F. Patel-Schneider, and B. Cuenca-Grau. OWL 2 Web Ontology Language Direct Semantics (Second Edition). W3C recommendation, W3C, Dec. 2012. URL `http://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/`.

[30] N. Ngo. Extended theoretical foundations for QUELO framework. Technical report, 2010.

[31] W.-l. Pan and D.-x. Liu. ORM-ML: XML schema for ORM metamodel. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 6, pages 55–59, 2010. ISBN 9781424455409. doi: 10.1109/ICCSIT.2010.5563603.

[32] W.-l. Pan and D.-x. Liu. ORM-ODM: Ontology Definition Metamodel for Object Role Modeling. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 6, pages 511–515. IEEE, 2010. ISBN 978-1-4244-5537-9. doi: 10.1109/ICCSIT.2010.5564692. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5564692`.

[33] W.-l. Pan and D.-x. Liu. Abstract Syntax of Object Role Modeling. In Z. Zeng and Y. Li, editors, *Fourth International Conference on Machine Vision*, volume 8350, Singapore, Jan. 2011. doi: 10.1117/12.920871. URL `http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.920871`.

[34] L. Perez-Beltrachini, C. Gardent, and E. Franconi. Incremental Query Generation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 183–191, Gothenburg, Sweden, 2014.

[35] M. Trevisan. A Graphical User Interface for Querytool. Technical report, KRDB, Bolzano, Italy, 2009.

[36] Y. Vassiliou. Null Values in Data Base Management a Denotational Semantics Approach. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, SIGMOD '79, pages 162–169, Boston, Massachusetts, 1979. ACM. ISBN 0-89791-001-X. doi: 10.1145/582095.582123.

[37] H. M. Wagih, D. S. ElZanfaly, and M. M. Kouta. Mapping Object Role Modeling 2 Schemes to OWL2 Ontologies. 2011.

[38] Wine.com. Catalog Objects, Sept. 2009. URL `https://api.wine.com/wiki/api-object-dictionary#_catalog_objects`.