Vrije Universiteit Brussel
Faculty of Sciences - Department of Computer Science
Semantic Technology and Applications Research Laboratory
Academic Year 2002 - 2003

Checking Consistency of Ontological Commitments by Employing an Inference Engine

Kurt Hoorne

Thesis submitted in partial fulfillment of the requirements for the degree of Licentiaat in de Informatica

Promoter: Prof. Dr. Robert Meersman

Abstract

An ontological commitment can be defined as the formalization of an ontology, thus consisting of the agreed terminology and the imposed constraints. To ensure that a commitment is consistent, i.e. mainly that an instantiation is possible and that there are no concepts that remain empty for every instantiation, some verification algorithms are required.
In this thesis the applicability of rule engines and inference engines to solve this problem with respect to the DOGMA ontology system is explored, thereby presenting a formal translation between binary ORM (expressed in the new Omega-RIDL ontology language) and the $\mathcal{SHIQ}$ description logic.
Furthermore we present the integration of the RACER inference engine into the DOGMA framework to automatically check the consistency of ontological commitments.

Acknowledgements

First of all, I would like to thank Prof. Dr. Robert Meersman for giving me the opportunity to have taken part in the exciting (though sometimes confusing) early phases of a promising ontology system and the foundation of a new language. That has been quite a unique experience.

I also want to thank Pieter Verheyden, Pieter De Leenheer, Jan De Bo, Peter Spyns and the other people at STARLab for their views and remarks that helped me to constitute this thesis.

A special thanks goes out to my parents for continually supporting me during the past tumultuous years.

Finally, peace out to my friends and all true soul musicians out there for keeping me going.

There is a certain rule that applies to all things in life: success demands commitment. This is generic knowledge to be treasured.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Originally a research domain in philosophy, the field of ontologies has gained growing attention from the IT community, especially with respect to the semantic web. In computer science, and the field of information systems in particular, the word ontology is defined as "terminology for a certain domain agreed on by domain experts". The conceptualization of the real world and the identification of all relevant relationships between them has proven to be a flexible and scalable way to build a correct and corresponding data model. In the past this conceptual modeling task was performed for one individual information system, but now researchers are taking it one step further by constructing ontologies to use them as a sort of generic, ideally universally agreed building blocks with which one can design better interoperable and/or more standardized applications. Basically, ontologies only consist of a terminology of concepts that are interrelated, which implies that, in order to meet the specifications of a certain system (one could even consider the case of adding semantics to web pages), constraints need to be supplied.

An ontological commitment according to Guarino et al [44] is stated as the following: *"Formalizing the ontological commitment of a logical language means offering a way to specify the intended meaning of its vocabulary by constraining the set of its models, giving explicit information about the intended nature of the modeling primitives used and their a priori relationships. In this sense, an ontological commitment is a mapping between a language and something which can be called an ontology."* This might sound a bit too abstract for some to directly apprehend, but it totally complies with what we shall define as an ontological commitment in the DOGMA system. It should already be obvious that without an ontological commitment, i.e. without any restrictions on the data, an ontology has no useful meaning for any application. Hence a formally defined connection between the conceptual, non-lexical model that is an ontology and the lexical nature of any software system is indispensable. By committing to an ontology applications can be made interoperable in case they (partly) share the same constraints or they can individually be made compliant to a certain standardized data model provided by some 'certified' ontology service.

Unfortunately, formulating ontological commitments is not a trivial occupation because inconsistencies can frequently occur: one can constrain an ontology

in such a way that it includes concepts and/or relationships that are proven to remain empty for every possible instantiation. This of course is caused by including redundant concepts and/or by imposing ill-defined, conflicting constraints inside a commitment. Normally, it is considered the responsibility of the modeling engineer to remove all redundancies and inconsistencies from the designed model. In general, this can become an extremely time-consuming task since commitments can be considerably large. Furthermore, there is no guarantee that all inconsistencies will be manually detected.
Instead of implementing a set of self-defined verification algorithms it seemed interesting to investigate the usability and compatibility of third-party reasoning services, which we shall explore in the following chapters.

1.1 Goal

The main goal of this thesis is to clarify what DOGMA ontological commitments exactly are, how they are built, and how they can be verified. To get a better understanding, we will talk about ontologies in the DOGMA system, the connections with description logics together with some related paradigms and existing tools that pursue the consistency checking of ontological commitments or constrained terminologies in general.

1.2 Overview

Chapter 2 gives an overall picture of the DOGMA system, the framework in which we shall define and apply ontological commitments. It also contains the introduction to Omega-RIDL, a new ontological commitment language.

In Chapter 3, we examine the use for DOGMA of two distinct sorts of systems called rule engines and inference engines, thereby discussing the fundamentals of description logics, which are prerequisite to understand the different mechanisms applied.

Chapter 4 describes the proposed integration of the RACER inference engine into DOGMA, including a translation mapping from binary ORM facts and constraints to the $\mathcal{SHIQ}$ description logic.

Finally, we will state our conclusions and summarize some relevant topics of future work.

Chapter 2

Ontological Commitments

2.1 Introduction

In this chapter we shall explain the DOGMA system, which is being developed at STARLab, focusing on the components related to ontological commitments. We will start off with a description of the global system itself, and then we will present a new language to express ontological commitments called Omega-RIDL, discussing its features.

2.2 The DOGMA System

The DOGMA (Developing Ontology-Guided Mediation of Applications) system embodies an expanding framework of tools with which one can develop, apply, process and convert ontologies. It currently stores ontologies by means of two layers: an ontology base for storing elementary facts with an commitment layer storing sets of constraints on top of it. These will now be explained a little more in detail, succeeded by some information on DOGMA modeler, an ontology editing and browsing tool, and ORM-ML, a markup language used to store and exchange ontological commitments. For a more exhaustive explanation of DOGMA we refer to [9] and [10].

2.2.1 Ontology Base

The ontology base stores a large collection of elementary and binary facts called lexons. Each lexon defines a relationship (labeled by a rolename) between two concept terms and is of the following form:

<context><term1><role><co-role><term2>

The co-role expresses the inverse relation between the two concept terms so it becomes redundant to add the 'inverse' lexon.

The context term tells in which context its bound fact is located, e.g. the domain of literature, politics, the fauna of southern Africa, . . .
Contexts render the ontology base more scalable and easier to reference. A wide range of dedicated literature is available and in the making, however contexts

of no real concern inside this topic as we will show later on.

Lexons define the static part of an ontology, e.g. "*Person has Age*" and "*Book has Author*" represent pieces of atomic knowledge that will not change over time. Note that subsumptions are stored in the ontology base as ordinary lexons using a generic subtyping "is-a" role[1].

N-ary roles are excluded from DOGMA because of their additional complexity and lesser granularity. An n-ary can always be expressed by using only binary roles (provided that a complete and correctly corresponding 'binary' conceptualization of the domain is available). This is why only binary facts are allowed.

All lexons, as well as all other parts of an ontology are stored onto the DOGMA server (to a relational database - see [10] for details), and because of their atomicity, it is not unthinkable that the global lexon base will consist of thousands (millions) of lexons.

2.2.2 Ontological Commitment Layer

Introduction

Defining an ontology without any rules has little or no use. In order to have an application making use of an ontology, constraints have to be added.
As opposed to lexons, constraints help express application-specific knowledge, e.g. "*Person has* **exactly one** *Age*" or "*Book has* **at least one** *Author*".

By storing these constraints in a separate layer, the degree of reusability and scalability significantly grows. This commitment layer, as it is called in DOGMA, forms the bridge between the ontology base and the applications that commit to it.

An example

To clarify the points we have previously discussed we now present an example of an ontological commitment in Figure 2.1, showing some constraints on the generic properties of a website.

Lexons define the graph structure of the ORM (Object-role modeling [1]) conceptual schema, depicted by the named ellipses and their connectors. Because lexons are exclusively binary, basic DOGMA ontological commitments are expressed in what we call binary ORM. The constraints are represented through all other symbols present in the picture.
The two subtypes of Person are shown here in ORM style with the black arrows, although inside DOGMA these are also lexons connecting a sub- and a supertype through a globally agreed "is-a" role. From this point on we will always present subtypes with this role instead of an arrow.

Also note that this is a fairly simple and small example directly created as an ORM diagram. In Section 2.3 we will introduce a more powerful formalism to formulate DOGMA commitments.

[1] This is maybe not definitive, since there is a lot of academic disagreement on the semantic difference between instantiation and subsumption of ontological concepts.

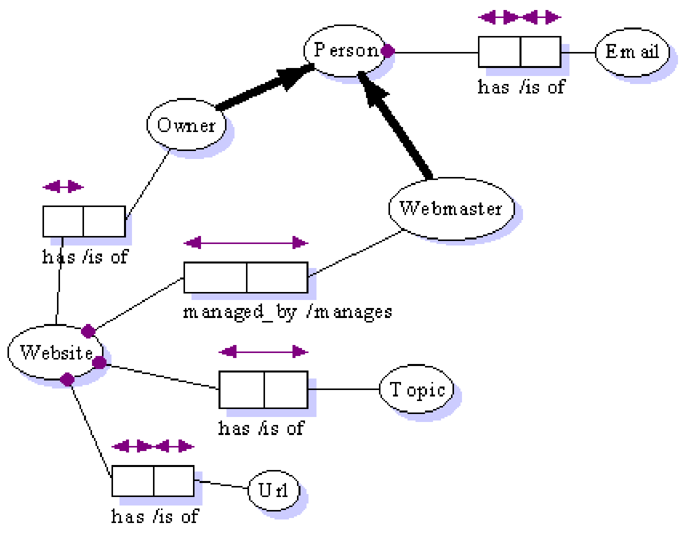


Figure 2.1: Example ontology

Definition

In the scope of the DOGMA system, an ontological commitment can be defined as *the formalization of an ontology base that is constrained according to some Universe of Discourse*. Thus a commitment contains in fact both a part of the ontology base (which is already a formalization in se) and a set of constraints.

The UoD mentioned will mostly be determined by the requirements of a certain application that commits to DOGMA. The formalization of an ontology is carried out by expressing it in ORM-ML, or with the new Omega-RIDL language.

Further on we'll show that there is absolutely a need to verify these formalizations, to have the guarantee that they are correct (read: consistent).

2.2.3 DOGMA Modeler

DOGMA Modeler is the client program connected to the DOGMA server designed to build, update and browse DOGMA ontologies. We give a short round-up of the features:

- New ontology bases and/or commitments are built by means of a graphical user interface to construct new lexons and/or add constraints.
- Lexon bases can be browsed and lexons can be drag-and-dropped into the graphical editor window to define new ontological commitments.
- Commitments can be expressed in pseudo-natural language by verbalizing all ORM facts and constraints.

- Some basic language-independent meta-schema is implemented to support multi-linguality: the same ontology can be built and viewed in different languages. This is still a hard issue to overcome [12], so for the moment DOGMA Modeler only incorporates a simple form of language-independence.
- Lexons and commitments are stored to the DOGMA server and commitments can also be saved in ORM-ML format (see next section).

The DOGMA Modeler tool is constantly being worked on and updated to meet the requirements of applications and projects that adopt the DOGMA methodology. Future implementation plans include support for alignment and merging of ontologies, support for Omega-RIDL constraints and queries, ...

2.2.4 ORM-ML

Up till now ontological commitments are stored in ORM-ML files. The ORM Markup Language [11] is an XML wrapper describing all referenced lexons from the ontology base as well as the constraints imposed on them. The choice of this standardized format makes it very suitable to exchange commitments over the world wide web.

Somewhat unfortunate, ORM-ML was designed to be machine-interpretable, so it becomes tedious to read as a human user. Despite its verbalization feature in the DOGMA Modeler tool, it is more appropriate to have a high-level conceptual language with which one can directly write down and read commitments. In that fashion, you have a formalism that is easy to learn and understand by people who are unfamiliar with the abstractions of traditional programming.

2.3 Omega-RIDL

As we have stated in the previous section, there is undoubtedly a need for an expressive and high-level constraint formalism. This is where Omega-RIDL fits in.

2.3.1 Origin

The principles of Omega-RIDL are not new. On the contrary, most of the ideas date from the late 70's and early 80's. We shall now give a brief outline of the foundations of conceptual information modeling.

ENALIM

ENALIM (Evolving NAtural Language Information Model) was one of the first proposals introducing the 'Idea-Bridge' method, where roles or 'ideas' as they were called, connected non-lexical entity types or concepts and where 'bridges' connected non-lexical with lexical entity types.

RIDL

Based on the above ideas came RIDL (Referential IDea Language/Laboratory) [2], a conceptual query and constraint language that was way ahead of its time, considering its very high-level nature.
In the beginning of the 80's a full implementation was developed called RIDL* [5], a tool for conceptually modeling a relational database. Among other, it contained a module called RIDL-A which performed basic consistency analysis of conceptual models.

Furthermore, within the RIDL Shell environment, a subsequent tool, it became possible to perform RIDL queries and updates. The RIDL syntax [2] allowed the user to textually formulate highly expressive conceptual queries by paths through the semantic network which was separately presented in a graphical way.

The new Omega-RIDL language, as we will explain further on, is very similar to RIDL syntax-wise, but it has modified semantics, adapted to the requirements of an ontology.

NIAM

During the construction of RIDL, the NIAM (Natural Information Analysis Method) [7] method was developed. It became a well-known standard for the natural language and AI community.

ORM

ORM (Object Role Modeling) [1] is the current incarnation of the ideas implemented in the languages explained above. Next to binary it also allows n-ary relationships between concepts and some new additions.

The ORM method provides a graphical constraint representation only (besides a few exceptions) and is primarily used to conceptually design relational databases.

2.3.2 Why define a new ontology language

The syntax used in DOGMA for storing ontologies was originally based on binary Object Role Modeling, but experience has shown that existing ORM constraints are not sufficient, e.g. the declaration of path equivalences (see next chapter) is unsupported. Also, because ORM was primarily meant to model relational databases, and not ontologies, specific syntax is missing to capture the description of an entire DOGMA commitment such as lexical mappings. This already implies that a higher expressivity must be attained by incorporating more powerful and complex constructs.

Furthermore, sometimes it is necessary to present the knowledge purely text-based, because it is not always appropriate to have a graphical representation. For example, one could be forced to develop an ontology that contains the precise contents of legal texts. In such a case, it becomes impossible to construct the ontology graphically.

In addition, an ontological commitment requires a standalone representation including all referenced lexons from the ontology base, even those that are not constrained (e.g. subsumption lexons). Therefore we need a complete syntax to express each component inside a commitment, enabling the users and engineers to have a coherent view.

Another issue is the notion of contexts, which are of great importance to combine several ontologies. Of course these are non-existent in ORM. To give a better idea, a DOGMA context could be seen as an RDF namespace. By nesting contexts an application can commit to multiple lexon bases, referencing the concepts it needs. They provide a useful structure for the alignment and merging of ontologies, a relatively new domain which many researchers are now working on.

Due to these shortcomings of ORM[2], the introduction of a new language was inevitable.

Instead of adopting an existing ontology language like DAML+OIL [49] and RDF(S) [50], we chose to build further on the ideas of ORM and RIDL which stand out for their simplicity in understanding. This is the power of Omega-RIDL: it is designed from a human or rather linguistic, not from a system point of view. It stands very close to natural language and is thus a very high-level paradigm.

Besides writing ontological commitments, Omega-RIDL can also be used to formulate conceptual queries, just like with the old RIDL and other conceptual query languages like ConQuer [8]. This of course implies an underlying mapping between the queried ontology and some information system such as a relational database. Note that, unless the fact that it is also meant as a conceptual query language, in this thesis we mainly focus on the constraint (read: commitment) part of the Omega-RIDL language.

[2]There has been a proposed constraint language called FORML (Formal ORM Language) [1], but it does not contain the expressivity we want to implement.

2.3.3 Some example commitments

Example 1 - with graphical ORM-equivalent

∗ ∗ ∗ preliminary Omega-RIDL commitment ∗ ∗ ∗
student_information.{
  Student has [is_of] **exactly one** Name
  Student has [is_of] **exactly one** Student_ID
  Student takes_daily [is_taken_daily_by] **at_least one** Train
  Student takes_daily [is_taken_daily_by] Train **excludes**
    Student occupies [is_occupied_by] **at_most 1** StudentRoom
  Student has [of] **at_least one** Hometown
  Train rides_to [is_visited_by] Town
  Hometown is-a [subsumes] Town
  StudentRoom is_located_at [houses] **exactly one** Address }

Table 2.1: Omega-RIDL commitment - Example 1

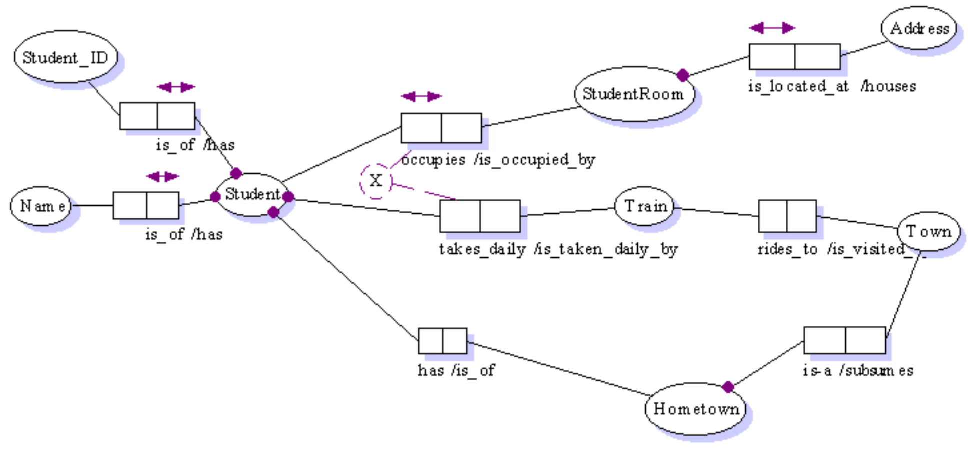


Figure 2.2: Omega-RIDL commitment - Example 1 - ORM view

Notice that there is an inconsistency inside this commitment: because the takes_daily role is mandatory and the occupies role and the former are exclusive, the occupies role and consequently the StudentRoom concept will be empty for every possible instantiation.

Example 2 - without graphical ORM-equivalent

This commitment cannot be completely represented through an ORM diagram. Only the terminology and its constraints could be drawn but it would not add much clarity in this case. Also note that we have left out the inverse roles for practical reasons, which is probably a better solution anyway for the end user.

```
∗ ∗ ∗ preliminary Omega-RIDL commitment ∗ ∗ ∗
student_information.{
if (Student has Degree of_type License
in Department "mathematics"
and Student enrolls_in Department "computer science")
then Student takes exactly one ShortenedCurriculum }
```

Table 2.2: Omega-RIDL commitment - Example 2

Omega-RIDL can be viewed as the extension of binary ORM with contexts, path equivalences, procedural constraints (for-each, conditional statements,...), etc.

2.3.4 Benefits

Let's recapitulate some benefits of this new ontological commitment (and query) language:

- It is much closer to natural language than DAML+OIL and other ontology languages. This implies rapid comprehension, enhanced usability, ...
- It is an extension of binary ORM, so the learning curb is kept low for people who are already familiar with ORM.
- There will be support for conceptual queries using a fairly simple syntax similar to the constraint syntax, making queries more transparent to the end user.
- One of the most important features will be the possibility to describe lexical mappings between applications and the DOGMA lexon base (see Section 2.4.1).

2.3.5 Usage

Maybe it is also convenient to explain who will make use of Omega-RIDL. We can distinguish three sorts of users:

Ontology engineers create ontology (lexon) bases and generic commitments that can be reused. This might involve several tools, e.g. the syntax of Omega-RIDL could be a good candidate to formalize natural language texts into lexons and constraints.

Commitment engineers formulate commitments in Omega-RIDL, connecting DOGMA lexons to application data.

End users can read the constraints (which should be easy to understand) and construct conceptual queries in case the respective ontology is used as a mediator for a database system.

Of course these are not fixed job assignments, they might overlap. We just mention this to exhibit the versatility of Omega-RIDL.

2.3.6 Consistency Problems

After an ontological commitment has been formulated, the question arises whether it is correctly constructed. In other words: since a commitment can be seen as a set of logical predicates, how can we be sure that we have made a consistent logical theory, devoid of conflicting and erroneous constraints?

First of all we can observe some minor errors that might have occurred during the construction of an ontology. These are mostly spelling mistakes and other ambiguities, e.g. roles that share the same name, but have another meaning. These are not actual inconsistencies and could be prevented for instance by including warnings for duplicate names. Therefore these errors are not examined any further here. The same holds for syntactical errors which can be detected by an Omega-RIDL parser. We also note that there is a difference between the consistency of an ORM diagram and the consistency of an ontological commitment.

In [32] four major consistency problems are recognized for a terminological knowledge representation system. Notice that Omega-RIDL is a terminological knowledge representation language of which we want to verify the consistency. Consequently there are four main problems that can be identified when examining ontological commitments:

1. **Knowledge Base Satisfiability**: is there a model (instantiation) possible for a certain commitment? (An application cannot commit if its ontological commitment is unsatisfiable.)
2. **Concept Satisfiability**: can a given concept C inside a commitment have at least one non-empty extension, i.e. can it be instantiated?
3. **Subsumption**: is a concept C more general than a concept D inside an ontological commitment, i.e. is there a (implicit) subsumption relationship between C and D?
4. **Instance Checking**: is an individual a an instance of a certain concept C inside a commitment?

It has been proven that all of these problems can be reduced to the Subsumption Problem and that they can all be solved by applying the appropriate reasoning algorithms.

In the next chapter we will show some systems implementing such algorithms and their use for DOGMA commitments, followed by a concrete proposed implementation presented in Chapter 4.

2.4 Notes

The introduction of Omega-RIDL to DOGMA will impose necessary changes on the system but it also brings new possibilities. We shall now discuss a few of these issues.

2.4.1 Bridging Commitments

The first topic is the 'syntactic depth' of commitments.

Next to the constraint part of ontological commitments (a collection of referenced lexons and imposed constraints), we can have 'application commitments', a set of constraints connecting concepts with objects (parameters, variables, table and column names, . . .) from an application committing to a certain ontology base. Application-specific constraints include terms for instances, support for concrete domains (assigning domain constraints to concepts, e.g. value ranges, integer, real, string, . . .), connections between ontological concepts and entities with properties inside the application, etc.

In analogy to the Idea-Bridge methodology of NIAM where a bridge signifies the connection between a non-lexical and a lexical (application-dependent) entity, these lexical mappings could be called bridging commitments. These commitments are useful in case a DOGMA ontology is used as the conceptual model or as a mediator for a relational database. Since these bridging constraints are very application-specific, they will be stored in separate bridging commitment layer, on top of their corresponding constraining commitments to acquire reusability of the latter. Although this is not formalized yet, it seems most logical to define a bridging commitment as a part of an ontological commitment since mapping rules are in fact also mere constraints. We emphasize that this solution is not definitive considering other proposals as in [12].
The syntax to express bridging commitments will be provided by the Omega-RIDL language, starting with a lexical mapping to SQL statements. Also note that there is no strict boundary between regular ORM constraints and the lexical constraints, e.g. a value range constraint in ORM could be considered a part of a bridging commitment.

An interesting feature of some of the old RIDL implementations was their so-called referability analysis. This was an algorithm that checked if all concepts in a certain conceptual schema were lexically referable so that there could be a mapping to a relational database schema[3]. Normally the goal of declaring a bridging commitment is to make a corresponding ontological commitment referable, so this feature could be reincarnated for DOGMA.

[3]This feature still exists in VisioModeler and other conceptual database modeling tools.

2.4.2 Omega-RIDL-ML

As Omega-RIDL will become the new standard to write ontological commitments, the question will arise if ORM-ML is still needed. It is possible that in the future there will be a transition to Omega-RIDL-ML, a yet to be defined markup language that is backward compatible[4] with ORM-ML. With Omega-RIDL-ML both the constrained lexons and lexical mappings could be stored and exchanged.

[4]It should be possible to port an ORM-ML commitment to an Omega-RIDL-ML file without any semantic loss.

Chapter 3

Inference Engines vs. Rule Engines

3.1 Introduction

It would be very convenient to have a theoretical mathematic foundation for our commitment language. That is why we now delve into the field of description logics, which provide a formal knowledge representation that has been thoroughly researched and documented.

A brief overview will be presented on the relations between binary ORM (formalized in the Omega-RIDL equivalent) and description logics and we'll discuss some well-known description logic formalisms. From there we shall investigate the applicability of description logics in both rule engines and inference engines. In the rest of this chapter we will cover the main differences between these two kinds of systems and their usefulness in terms of our research.

3.2 Description Logics

3.2.1 Introduction

Description logics (DL) are based on a subset of function-free First Order Logic[1]. They can act as a form of knowledge representation within a proof-theoretic framework. Because of their unambiguous and restricted nature description logics are perfectly suited to implement reasoning algorithms as one can find inside inference engines.

An important characteristic to emphasize at first is that all DL constructs, including constraints, define a set of instances related to one or more concepts (see Appendix A for more information). This means the same syntax can be used to define constraints and to formulate queries, a feature that significantly enhances the simplicity.

There are very expressive description logics in existence such as $\mathcal{DLR}$, but in order to have an acceptable performance for reasoning on DL knowledge bases

[1]Except if support for transitive closures (a logic fixpoint constructor) is also added to the language.

and decidable algorithms, we are forced to restrict the number of constructs by restricting to a small but expressive language such as $\mathcal{AL}^*$ (see next sections). By doing so, we take the risk of losing some of the expressiveness inherent to Omega-RIDL.

We shall now present the relationship between Description Logic and Omega-RIDL, on which we will build further in the following sections.

3.2.2 Mapping Omega-RIDL to Description Logic

To reason with ontological commitment statements (constraint code), we shall fall back on reasoning with description logics. To accomplish this, we first need to syntactically connect the two forms of knowledge representation.

We could have shown a proposed mapping between a subset of Omega-RIDL and description logics, but this is of not much use since there are no systems implementing the whole of constructors possible in description logics. It is more sensible to translate to an applied description logic, as the one we shall discuss in the next section.

Not every construct in Omega-RIDL will have a semantic counterpart in description logic, and in order to make use of decidable and fast reasoning algorithms for our ontological commitments, we will have to restrict to a minimized but still sufficiently expressive description logic. Thus there will inevitably be a minimized subset of features in Omega-RIDL that will be lost while translating.

DOGMA Contexts in Description Logic

Lexons in the DOGMA system are all bound to a certain context. Contexts could be considered lambda-environments (Church et al). For the mapping of the context constructor we suggest the following: since lexical scopes will normally (but not necessarily - see [45]) imply higher order reasoning and likewise consistency checking, we should avoid it by applying some modifications before translating Omega-RIDL into DL.

If the context is only used in at the beginning of a commitment, it can be omitted. In case nested contexts are present inside a commitment, e.g. to express the equivalence of two possibly synonymic concepts from different contexts, we'll rename the participating items:

$context_X.C_i$ **is-equivalent-to** $context_Y.C_j$

can be mapped to

$(\text{``}context_X.C_i\text{''} \subseteq \text{``}context_Y.C_j\text{''}\ AND\ \text{``}context_Y.C_j\text{''} \subseteq \text{``}context_X.C_i\text{''})$.

In words: the contexts are recursively concatenated as strings to all their bound concepts and roles to get as endresult a translation in description logic free from contexts.

3.2.3 $\mathcal{SHIQ}$ and $\mathcal{SHOIQ}$

Of course we want to know to what description logic Omega-RIDL is supposedly to be equivalent. Table 3.1 shows an overview of the naming of DL languages according to the constructs they support as explained in [31]. For coherence reasons we have denoted concepts by C_i and role(name)s by R_i. We use the notation a_i to depict instances of concepts. This naming convention is kept throughout the whole chapter.

Construct	**DL Syntax**	**Language**
concept	C	$\mathcal{AL}^*$
role name	R	
existential quantification	$\exists R$	
intersection	$C_1 \cap C_2$	
value restriction	$\forall R.C$	
top $\equiv$ transitive closure (‘thing’)	$\top$	
bottom $\equiv$ empty set (‘nothing’)	$\bot$	
negation	$\neg C$	$\mathcal{C}$
disjunction	$C_1 \cup C_2$	$\mathcal{U}$
existential restriction	$\exists R.C$	$\mathcal{E}$
number restriction	$(\leqslant n\ R)$	$\mathcal{N}$
	$(\geqslant n\ R)$	
	$(= n\ R)$	
collection of individuals	$a_1, a_2, ..., a_n$	$\mathcal{O}$
role hierarchy	$R_1 \subseteq R_2$	$\mathcal{H}$
inverse role	R^-	$\mathcal{I}$
quantified number restriction	$(\leqslant n\ R.C)$	$\mathcal{Q}$
	$(\geqslant n\ R.C)$	
	$(= n\ R.C)$	
role conjunction	$R_1 \cap R_2$	$\mathcal{R}$
transitive roles	$R \subseteq R^+$	$\mathcal{R}^+$
functional roles (features)	$R : \{state_1, state2, \ldots\}$	f

Table 3.1: Classification of Description Logic Languages

$\mathcal{AL}^*$ is the name of one of the best-known elementary description logics. There are many possible naming combinations for the extension of $\mathcal{AL}^*$, e.g. the language containing negation and inverse and functional roles is called $\mathcal{ALCI}_f$. Some clarifications:

- Transitive roles are roles that can be a subset of their respective transitive closure, e.g. the role *has_parent*: *son has_parent father* and *father has_parent grandfather* and so on. For the ORM people: these are equivalent to transitive binary ring constraints. (R^+ depicts the transitive closure of R)

- Functional roles or features are roles that can have only a value as a filler instead of concepts, e.g. the feature *has_color*: *banana has_color #yellow* with *#yellow* describing a state/value, not a concept.

The DL $\mathcal{ALC}_{\mathcal{R}^+}$, i.e. $\mathcal{ALC}$ extended with transitive roles, is also called $\mathcal{S}$. Hence you can form the languages $\mathcal{SHIN}$, $\mathcal{SHIQ}$, ...

To define a mapping for Omega-RIDL to description logics, we will have to apply several tricks to get a good solution. In many cases, translation will be possible, in some it will be impossible. A few examples:

- Omega-RIDL (just like ORM) does not explicitly support role hierarchies, leaving us with no real use for the $\mathcal{H}$ description logic role constructor.
- Mandatoriness of roles is translated into a qualified number restriction: a mandatory role must connect each instance of a concept with at least one instance of the related concept.
- Role exclusion constraints (depicted in ORM by the symbols $\otimes$ and the combination of $\otimes$ and $\odot$) are not directly supported in $\mathcal{SHIQ}$. We can sidestep this by defining an exclusion constraint (a disjunction) on the concepts that define the range of the respective roles. In case it is a total exclusion constraint, we delegate the mandatory constraints to all participating concepts and define a disjunction between the concepts participating in the disjoint roles.

After a first exhaustive examination of our mapping problem, we can already identify at least two ORM/Omega-RIDL constraint types that cannot be translated with any of the constructors in Table 3.1:

1. External role uniqueness constraints
2. M:n role uniqueness constraints

In the next chapter we'll give a concrete translation into $\mathcal{SHIQ}$, but we can already state that the first version of the commitment language Omega-RIDL will be equivalent to $\mathcal{SIQ}$ extended with constructs for the two aforementioned features. The $\mathcal{H}$ is omitted because role hierarchies don't exist in ORM and Omega-RIDL, so we don't need the constructor. Later on, when statements related to instances are added, it will upgrade to an extension of $\mathcal{SOIQ}$.

We already know [34] that DAML+OIL is equivalent to $\mathcal{SHOIQ}$[2]. This means Omega-RIDL and DAML+OIL are very alike semantically, although there is a large difference syntactically seen. Nonetheless it will be possible to employ $\mathcal{SHOIQ}$ reasoning services to analyze DOGMA ontological commitments, as will be explained in the next chapter.

[2]In fact, DAML-OIL also supports the assignment of concrete datatypes to its concepts. Its DL equivalent (without inverse roles) is called $\mathcal{SHOQ}$(D) [40]

3.3 Description Logic Programming

Description Logic Programming (DLP) [39] is a relatively new domain, which is based on the semantic intersection of Description Logic and Logic Programming (LP). The goal of this framework is to use the inferencing power of a LP system for a knowledge base defined in some Description Logic. To reach this, the respective LP language has to be restricted to the semantic equivalence of DL (a subset of First Order Logic) so that there can be a lossless mapping in both directions. In practice this means a DL knowledge base is translated in LP facts and rules, derived facts are computed, and the whole gets retranslated to an extended DL knowledge base.

There are several DLP subdomains according to the studied description logic, e.g. the mapping of DL strictly to Horn rules which leads to knowledge representation languages using exclusively Horn rules [36].

We will show an example of Description Logic Programming in section 3.7.4.

3.4 Constraint Logic Programming

Another paradigm related to Logic Programming is Constraint Logic Programming (CLP) [37]. CLP is the fusion of constraint systems and Logic Programming systems that are both built on declarative frameworks based on relations and backtracking search. It has been developed to achieve a performant solution to problems concerning planning, configuration, scheduling, etc.
Most CLP languages are extensions of the Prolog language (see section 3.7.2) and popular systems include CHIP, Prolog III, ECLiPSe, ...

CLP systems are closely related to rule engines, but they implement a different approach to solve a problem [38]. CLP constraints are written down in constraint tables (and additional rules), while LP rules are of the form "if-then". If a new fact changes one or more constraints, a constraint-based system is rapidly updated. In contrast, it is very likely that some rules will have to be deleted and retracted and that other rules will have to be added in a rule-based system. These rule updates are expensive operations, which implies that the usage of rules for genuine constraint-related problems is fairly inadequate.

Although its name makes it tempting to explore for DOGMA, it should be obvious that Constraint Logic Programming is not suited for processing ontological commitments.

3.5 A-Box vs. T-Box Reasoning

There are two other important concepts frequently referred to in description logic literature that we need to mention: T-Box and A-Box reasoning.

- T-Box reasoning comes down to computing information for a given Terminology of concepts with their respective relations etc. Hence a T-Box for an ontological commitment consists of all concepts with all their respective roles, subsumptions and constraints.
- With A-Box reasoning Assertional knowledge is processed, i.e. instances of concepts are examined and used to assert facts (e.g. "*Mary has_child Leon*", "Car_1 *is_painted black*", ...).
  We recall that the $\mathcal{O}$ description logic construct provides syntax to make A-Box statements that associate instances with values.

This distinction is also present inside the ORM view, which separates the conceptual model and its instantiation/sample populations.
Examples of T-Box and A-Box statements will arise in this and the following chapter.

3.6 Open World Assumption

Most description logic systems employ the Open World Assumption, meaning every fact that is not true inside a given T-Box/A-Box terminology is believed not to be false and remains unknown until a newly inserted fact states otherwise.

In contrast, rule engines and most database systems for example assume a Closed World, eliminating a large number of queries and thereby increasing their inferencing performance.

When considering ontologies, in general an Open World is assumed, primarily due to the dynamic and open character of the semantic web. This will also be the case for ontological commitments in the DOGMA system, unless of course an ontology-mediated database is examined.

3.7 Rule Engines

3.7.1 Basics

Rule engines are programs that were also known as expert systems which were very popular during the 80's in the field of Artificial Intelligence. By means of pattern recognition and the firing of rules they compute derived facts for a given knowledge base. The rules are in fact nothing more than sophisticated if-then statements, executed in a fast, predefined and non-deterministic way
(A rule is immediately fired whenever its conditions are satisfied, while embedded rule priorities ensure rule conflict resolution.).

Since rule engines are systems implementing the methods of Description Logic Programming we were interested in which way they could contribute to the DOGMA system.

Based on their core architecture, there are two major families of rule engines: Prolog- and Rete-based [15] systems. Both stand for fast inferencing algorithms. Another distinction can also be made between forward chaining and backward chaining engines, depending on how rules are fired. We do not go into this in detail because it is not important in the scope of our research.

We will now discuss some rule-based systems.

3.7.2 Prolog

Probably the most well-known Logic Programming language is Prolog, which is based on a subset of First Order Logic. A Prolog system can be used as a rule engine. As in most other (Lisp-based) logic systems, expressions are written in prefix notation. To give an idea of what Prolog statements look like, we show the syntax to declare facts and rules:

- Facts are declared using a straightforward notation, e.g. to express that an instance called martin is parent of charlie we write: $parent(martin, charlie)$.
- Rules are expressed by first stating the conclusion, followed by the conditions to derive it: $grandparent(x, z) :\!- \, parent(x, y), parent(y, z)$.

Note that Prolog is not a purely declarative language since the order of assertions determines the execution process.

3.7.3 CLIPS

The CLIPS (C Language Integrated Production System) expert system [13] was developed at NASA, to be used as a rapid prototyping tool, to develop production prototypes and production applications. It is Rete-based and can represent both facts and rules as well as objects.

3.7.4 JESS

Since CLIPS is solely a C-distribution, it has been rebuilt for the Java platform under the name JESS (Java Expert System Shell). JESS [14] is an open-source project for the academic community. It is very similar to the CLIPS engine, but only implements a subsection of its functions and has some extra Java-specific features.

We have done some experiments in JESS to explore its capabilities. The first use we found is poor T-Box inferencing: it is possible to design a set of rules defining the subsumptions and regular relationships (roles) between concepts. After these rules have been fired, implicit subsumptions will be inferenced as new facts.

Another important finding is that JESS, like most other rule engines, does not include consistency checks for rules. Even worse, conflicting or overlapping rules merely overwrite each other, so there is no guarantee that the translated constraints of a certain knowledge base will impose their original restrictions on the concepts.
We explored the possibility of constructing rules to explicitly impose the consistency of translated binary ORM constraints, but that did not work out due to the high complexity.

The only real benefit of a rule engine with respect to ontological commitments is situated in basic A-Box reasoning. A simple instance validation algorithm can be implemented to compare instances to the knowledge base (terminology) of a certain ontology. This happens through the process of pattern binding. Consequently, we could have used JESS as A-Box reasoner, but as this was not our main goal, we abandoned it.

As you can see below, the syntax of JESS is a bit different from Prolog:

- Facts are written down in a similar fashion:
  `(assert (`*parent martin charlie*`))`
- JESS rules have their conditionals in front of their goal and they all need a unique identifier:
  `(defrule` *grandparent_rule* `(`*parent ?x ?y*`)(`*parent ?y ?z*`)`
  `=>(`*grandparent ?x ?z*`))`

To give an idea of how to implement this A-Box reasoning, we have translated some Omega-RIDL statements into JESS rules. This incomplete translation is shown in Table 3.2. As you can see, conceptual constraints have to be transformed into if-then rules to be accepted by the system.
Unfortunately, many constraints, such as cardinality restrictions, cannot be translated in a direct way (or even not at all), so this is definitely not the best solution to deal with ontological commitments.

Omega-RIDL	**JESS rule**
C_2 **is-a** C_1 $(subtyping)$	`(defrule` $subsumption$ $(subsumed\_by\ ?C_2\ ?C_1)$ $(instance\_of\ ?x\ ?C_2)$ => $(instance\_of\ ?x\ ?C_1))$
C_1 R **one-or-more** C_2	`(defrule` $mandatory\_role$ $(C_1\ ?x)$ => $(R\ ?x\ ?y))$
$C_1\ R\ [R^{-1}]\ C_2$ $(inverse\ or\ co-role)$	`(defrule` $inverse\_role$ $(R\ ?x\ ?y)$ => $(R^{-1}\ ?y\ ?x))$
R **is-transitive** $(transitive\ role)$	`(defrule` $transitive\_role$ $(R\ ?x\ ?y)$ $(R\ ?y\ ?z)$ => $(R\ ?x\ ?z))$
instance binding to concept	`(assert` $(instance\_of\ a_1\ C_1))$
instance binding to role	`(assert` $(R_i\ a_1\ a_2))$

Table 3.2: Omega-RIDL translated into JESS-rules

3.7.5 Other Rule Engines

Next to open source systems, there are a lot of commercial rule engines available on the market, such as the Authorete engine from Haley [16]. The main differences lie in performance, user interfaces and extra features. Rule engines are primarily used to maintain, update and execute business rules.

3.7.6 Summary

Rule engines are only useful for processing ontological commitments if they are employed for Description Logic Programming purposes, in particular as basic A-Box reasoners. In general there is no consistency checking for rules present, and it is impossible to have a translation for all kinds of constraints.

Consequently, we have left out rule engines from our current research.

3.8 Inference Engines

3.8.1 Basics

As opposed to a rule engine, an inference engine is a tool that does not accept rules but exclusively description logic statements. Constraints are not to be expressed as rules and can be left declarative so that they can be identified by means of their labels. In other words: you only need to feed an inference engine purely with descriptive knowledge, because the semantics of constraints are handled internally. Inference engines can be seen as theorem provers for a certain Description Logic that decide whether a given knowledge representation is valid or not.
Currently we have encountered two advanced inference engines that are publicly available: FaCT and RACER.

3.8.2 FaCT

FaCT (Fast Classification of Terminologies) [22] is one of the rare non-commercial inference engines. It is designed to reason with DAML-OIL terminologies, implementing advanced tableau algorithms for $\mathcal{SHOIQ}$.

Despite the fact that it is free, we chose not to use it to verify ontological commitments. The two major drawbacks were its very concise documentation and its CORBA interface, which makes it impossible to easily incorporate into a Java-based system such as DOGMA. Furthermore, the public version only supports T-Box reasoning and does not implement cardinality restrictions.

Recently this engine has been improved and put on the commercial market as the CEREBRA engine [17].

3.8.3 RACER

Similar to FaCT there is RACER (Renamed ABox and Concept Expression Reasoner) [25], a T-Box and A-Box reasoner for the $\mathcal{SHOIQ}$ logic implemented in Lisp. It even has limited support for concrete domains, meaning you can impose rational and integer domain constraints upon concepts.

There is also an open source Java API available to connect and communicate with a RACER server, as well as a graphical interface client program to examine a T-Box/A-Box terminology.

All of RACER's features make it very suitable to integrate within the DOGMA system, and this is what we will propose in the next chapter, together with a translation mapping from Omega-RIDL into RACER syntax (equivalent to $\mathcal{SHOIQ}$).

To our regret, we discovered RACER much too late, so there was not enough time to implement the ideas supplied in the following chapter.

3.8.4 Tableau Algorithms

The algorithms applied in Description Logic reasoners such as FaCT and RACER are mathematically founded in the domain of tableau calculus.

Tableau algorithms check the consistency of a terminology (T-Box and also A-Box) by instantiating concepts and/or roles and propagating these throughout all connections until a contradiction is found. If no contradiction is detected, of course the terminology is consistent.
To achieve better performance, a whole range of optimizations can be included, according to the implementation.

For more information about tableau algorithms for DL reasoning we refer to [34] and [35].

3.8.5 Summary

Inference engines provide a solution to our problem of verifying the consistency of DOGMA commitments. Of course, there would have to be a mapping between Omega-RIDL statements and the Description Logic implemented by the inference engine, which is always possible if there are no fundamental differences in semantic expressiveness.

3.9 Conclusion

In the scope of ontological commitments, rule engines are of not much use since they are mere derivers of facts. It is possible to translate a commitment to a set of rules, but they will not be checked for consistency by the rule engine. Overlapping rules will not be detected and will be imperatively fired. The re-translation of the rules and facts will provide an updated ontology with possibly additional inferred facts, but it cannot be guaranteed that these will be consistent with the originally constructed ontological commitment.

Inference engines however provide a solution to detect inconsistencies in a predefined format, in accordance with a certain description logic. The RACER inference engine is such an advanced reasoner for the $\mathcal{SHIQ}$ description logic. As we already have suggested that commitments written in Omega-RIDL can be mapped to $\mathcal{SHIQ}$ (with some loss of expressiveness), we can also map Omega-RIDL to input for an inference engine. This way, we can leave the consistency checking of ontological commitments to the reasoning algorithms inside an inference engine.

In the next chapter we shall propose the integration of the RACER engine into the DOGMA system, including a proper translation of binary ORM constraints into $\mathcal{SHIQ}$ and a proof of concept.

Chapter 4

Verification of Ontological Commitments

4.1 Problem Definition

4.1.1 Introduction

Adopting the T-Box/A-Box distinction, we can define the description logic equivalent of an ontological commitment as a pair $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is a T-Box and $\mathcal{A}$ represents an A-Box. An interpretation $\mathcal{I}$ is a model for Σ if it is a model for both $\mathcal{T}$ and $\mathcal{A}$. With this knowledge, let's recapture the problems stated in Section 2.3.6:

1. **Knowledge Base Satisfiability**: a commitment Σ is satisfiable if it has a model
2. **Concept Satisfiability**: a concept C is satisfiable w.r.t. Σ if there exists a model $\mathcal{I}$ of Σ such that $C^{\mathcal{I}} \neq \emptyset$
3. **Subsumption**: C is subsumed by D w.r.t. Σ, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of Σ
4. **Instance Checking**: a is an instance of C, written $\Sigma \models C(a)$, if the assertion $C(a)$ is satisfied in every model of Σ

With this formalization we have unambiguously defined the consistency[1] problems for commitments, and these are exactly the problems that can be tackled by an inference engine such as RACER.

Omega-RIDL commitments will be stored for reference and reuse. Of course it would be impractical to store incorrect or inconsistent commitments. Thus we propose to integrate the RACER $\mathcal{SHOIQ}$ reasoner in the DOGMA system to impose a consistency verification before compiling ontological commitments to a specified storage format.

[1]Sometimes T-Box consistency is called coherence, which may be a better term.

4.1.2 Omega-RIDL Compiler

Currently being developed, the Omega-RIDL compiler is a new component of the DOGMA system that will parse Omega-RIDL commitments, verify them for consistency, and then translate them into a yet to be defined file format. This will most probably be an XML-compliant format, based on a Omega-RIDL DTD.

4.1.3 DOGMA Contexts

We recall that contexts are left out from the reasoning process by applying the modifications explained in section 3.2.2.
The applied method is totally legal since the syntax of contexts adds no extra semantic restrictions on an Omega-RIDL commitment.

Reasoning on DOGMA contexts will probably become possible, but this is a problem situated on a whole other level than that of ontological commitments.

4.2 Existing Ontological Consistency Checkers

We already described the RIDL-A module inside RIDL* [3] which analyzed conceptual schemas by treating them as semantic networks.
Of course there are also some more recent applications that verify the consistency of ontologies.

4.2.1 ConsVISor

Besides FaCT and RACER, we have found one tool that explicitly claims to check the consistency of ontologies: ConsVISor [18].
ConsVISor is designed to verify the consistency of ontologies related to the semantic web.

Unfortunately this is a rather poor tool, using a Prolog rule engine and SNARK [47], a resolution-and-paramodulation theorem prover for first-order logic with equality. We have already shown the disadvantages of this in the preceeding chapter.

4.2.2 DISCOVER

The DISCOVER tool described in [48] is used to detect anomalies between an ontology and a knowledge base that commits to it. DISCOVER incorporates the COVER tool which has its own COVER Rule Language (CRL) applied on top of terminologies defined in the MOVES ontology language.

We found it rather unpractical due to its complexity and its different approach to ontologies.

4.2.3 i-com

i-com [19] is a tool for checking the consistency of (extended) EER diagrams that connects to the FaCT inference engine via its CORBA interface. It implements a complete inferencing solution to check the consistency and to make deductions of the conceptual models.

4.3 Consistency Checking with the RACER Inference Engine

4.3.1 RACER Basics

Like every description logic system, RACER is built on a concept language, defining all legal concept terms, and an elementary role language. The corresponding BNF is shown in Table 4.1.

$C \rightarrow$	CN	|
	`*top*`	|
	`*bottom*`	|
	(`not` C)	|
	(`and` $C_1 \ldots Cn$)	|
	(`or` $C_1 \ldots Cn$)	|
	(`some` R C)	|
	(`all` R C)	|
	(`at-least` n R)	|
	(`at-most` n R)	|
	(`exactly` n R)	|
	(`at-least` n R C)	|
	(`at-most` n R C)	|
	(`exactly` n R C)	|
	CDC	
$R \rightarrow$	RN	|
	(`inv` RN)	

Table 4.1: RACER Concept and Role Constructors

The concept `*top*` signifies the most general concept of a T-Box. Likewise, the `*bottom*` concept denotes the inconsistent, incoherent or unsatisfiable concept.

RACER clearly distincts between its T-Box and A-Box component by separating the functionalities for both components. Furthermore, we recognize that it is in fact a reasoner for the $\mathcal{SHOIQ}(\mathcal{D})$ description logic. The D stands for a limited support of a feature called Concrete Domains, as explained in Section 4.3.3.

RACER employs the Open World Assumption and Unique Name Assumption, meaning that all instance names refer to a different element in the UoD.

4.3.2 Integration into DOGMA

Firstly, the RACER server program for Windows (or UNIX - both are available) is started by running the executable, hereby promoting the respective computer system in the network to a RACER server. This server has three interfaces: a file-based, a HTTP-based and a TCP Socket-based one.

A full Java API has been developed to communicate with the RACER server via the TCP/IP protocol. With this JRACER API terminologies can be sent to the server and checked for consistency.

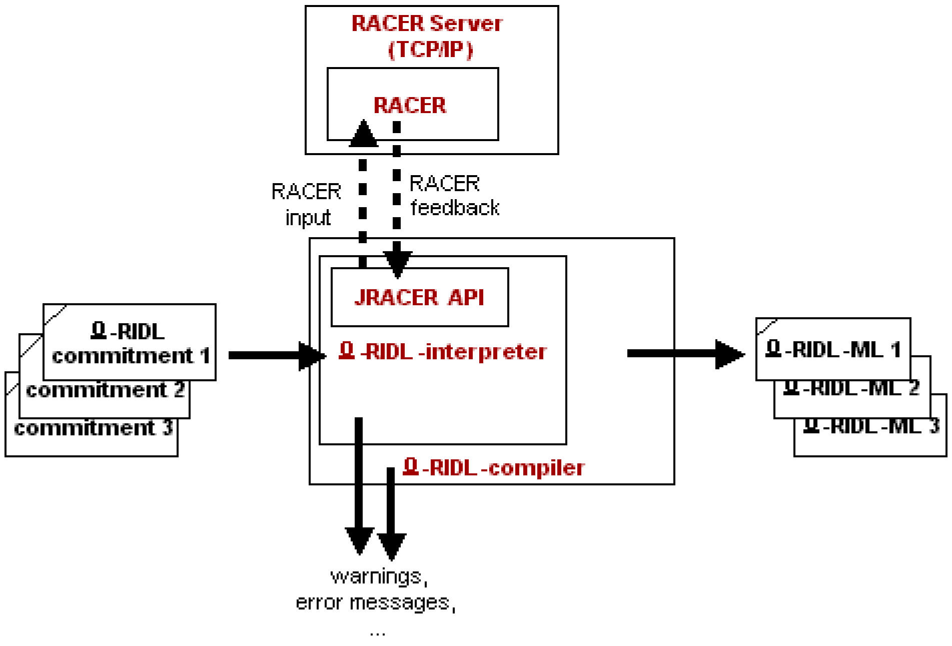


Figure 4.1: Integration of RACER into DOGMA

The integration of RACER into DOGMA is fairly simple as shown in Figure 4.1: a pc in the network is used as RACER server, and its (constant) IP address is stored inside the Omega-RIDL interpreter/compiler to connect with the server via the JRACER API.

The consistency checking of an ontological commitment is equivalently easy to implement: Omega-RIDL compiler feeds RACER the translation of a parsed commitment through the JRACER Java API, consistency is verified by sending the appropriate function calls (see Section 4.3.4) to the server, results are sent back to the compiler and the commitment is compiled into the wanted output format. Of course inconsistent commitments will not be compiled: an error message will be shown instead.
Note that the same dataflow applies if one only considers the interpretation of an Omega-RIDL commitment; in that case the compilation is left out from the process.

4.3.3 Omega-RIDL-to-RACER Mapper

First of all, the parsetree of a parsed Omega-RIDL commitment will be translated into RACER syntax so it can be used as input to reason on.

This is a preliminary version, seen Omega-RIDL is a language that is still being constructed. This means the constructs shown here can differ in the final version of Omega-RIDL. In general, the old RIDL syntax is usually adopted. Note that it will be possible to formulate the same constraint in several ways in Omega-RIDL. Furthermore, RACER also in many cases supplies more than one way to express a description logic feature (we refer to the official RACER manual [27] for all 'syntactic sugar' statements). For a formal translation of binary ORM into $\mathcal{SIQ}$ we refer to Appendix B.

Omega-RIDL	**RACER input**
$Concept_1$ *role* [*co-role*] $Concept_2$	`(signature` `:atomic-concepts` ($Concept_1$ $Concept_2$) ...) `(define-primitive-role` *role*) `(inverse` *role co-role*)
$Concept_2$ **is-a [subsumes]** $Concept_1$	`(implies` $Concept_2$ $Concept_1$)
$Concept_1$, $Concept_2$, ... **are-disjoint** *(disjointness of concepts)*	`(disjoint` $Concept_1$ $Concept_2$ ...)
$Concept_1$ *role* **one** $Concept_2$	`(implies` $Concept_1$ `(exactly` 1 *role* $Concept_2$))
$Concept_1$ *role* **at-most-one** $Concept_2$	`(implies` $Concept_1$ `(at-most` 1 *role* $Concept_2$))
$Concept_1$ *role* **one-or-more** $Concept_2$	`(implies` $Concept_1$ `(at-least` 1 *role* $Concept_2$))
$Concept_1$ *role* **at-most** *n* $Concept_2$	`(implies` $Concept_1$ `(at-most` *n* *role* $Concept_2$))
$Concept_1$ *role* **at-least** *n* $Concept_2$	`(implies` $Concept_1$ `(at-least` *n* *role* $Concept_2$))
$Concept_1$ *role* **exactly** *n* $Concept_2$	`(implies` $Concept_1$ `(exactly` *n* *role* $Concept_2$))
$Concept_1$ $role_1$ $Concept_2$ **implies** $Concept_1$ $role_2$ $Concept_3$ *(role subset constraint)*	`(implies` `(and` $Concept_1$ `(some` $role_1$ $concept_2$)) `(and` $Concept_1$ `(some` $role_2$ $concept_3$)))
$Concept_1$ $role_1$ $Concept_2$ **equals** $Concept_1$ $role_2$ $Concept_3$ *(role equality constraint)*	`(equivalent` `(and` $Concept_1$ `(some` $role_1$ $concept_2$)) `(and` $Concept_1$ `(some` $role_2$ $concept_3$)))
$Concept_1$ $role_1$ $Concept_2$ **excludes** $Concept_1$ $role_2$ $Concept_3$ *(role exclusion)*	`(equivalent` "$Concept_1 role_1$" `(some` $role_1$ $Concept_2$)) `(equivalent` "$Concept_1 role_2$" `(some` $role_2$ $Concept_3$)) `(disjoint` "$Concept_1 role_1$" "$Concept_1 role_2$")
$Concept_1$ $role_1$ $Concept_2$ **xor** $Concept_1$ $role_2$ $Concept_3$ *(mandatory role exclusion)*	`(equivalent` "$C_1 role_1$" `(at-least` 1 $role_1$ C_2)) `(equivalentt` "$C_1 role_2$" `(at-least` 1 $role_2$ C_4)) `(disjoint` "$C_1 role_1$" "$C_1 role_2$")
$Concept_i$ **is-restricted-to** { *j..k* } *(value constraint)*	`(signature ...` `:attributes ((integer|real` $attrib_1$)) ...) `(equivalent` $Concept_1$ `(and (min` $attrib_1$ *j*) `(max` $attrib_1$ *k*)))
at-most #*n* **of** $Concept_1$ **are-allowed** *(object frequency constraint)*	for all mandatory roles $role_m$ on $Concept_1$: `(implies` $Concept_m$ `(at-most` #*n* `(inv` $role_m$) $Concept_1$))
$role_i$ **is-transitive**	`(define-primitive-role` $role_i$ `:transitive t))`
$role_i$ **is-symmetric**	`(define-primitive-role` $role_i$ `:symmetric t))`
$Concept_1$ $role_x$ $Concept_2$ $role_y$ $Concept_3$ ... *(path-expression)*	`(and` $Concept_1$ `(some` $role_x$ `(and` $Concept_2$ `(some` $role_y$ `(and` $Concept_3$ (...))))
*concept-exp*1 **is-equivalent-to** *concept-exp*2 *(path-equivalence)*	`(equivalent` *concept-exp*1 *concept-exp*2)
if *concept-exp*1 **then** *concept-exp*2 *(terminological if-then)*	`(implies` *concept-exp*1 *concept-exp*2)

Table 4.2: T-Box translation of Omega-RIDL to RACER input

Also, for demonstration purposes the different translations are shown here separately, but usually a T-Box is entered in RACER as one whole of concept and role declarations and constructs. This whole is called the signature of a terminology. Apart from the declaration of atomic concepts, every constructor can be added after defining the signature by a calling a separate macro or function.

Let us now clarify the above translations in the following points:

Lexons

Lexons are elementary facts, defining a relationship named by a rolename that connects two elementary or atomic facts, including the rolename of the inverse relationship. In RACER, atomic concepts have to be defined in the signature of a T-Box: there is no independent constructor for atomic concepts. Domain and range can be included to improve performance, but this is allowed only if the role inside the lexon has a unique name.

Subtyping

In contrast to ORM and description logic, subsumptions in Omega-RIDL are expressed by means of a regular role named "is-a" instead of a special construct. For now we have assumed that each and every "is-a" role defines a complete subsumption, implying that a subtype inherits all of its supertype's relationships and other properties. Maybe in the future it could become possible to differentiate the "is-a"'s by adding the necessary commitment syntax and interpretation mappings.

Note again that RACER clearly distinguishes between instantiation (an A-Box assertion) and subsumption (a T-Box assertion).

Mandatory roles

Due to their semantics and graphical notation in ORM, there is a small catch when mandatory roles are verbalized and translated. The mandatory constraint for a concept on a role is expressed by defining a qualified number restriction on the connected concept inside the same lexon.

Transitivity, reflexivity & symmetry of roles

RACER explicitly supports transitive, reflexive and symmetric roles (cfr. ring constraints in ORM)[2]. These respective properties can be specified by extending the role definition by using some of the optional parameters:
(`define-primitive-role` role (`transitive` *nil*) (`reflexive` *nil*) (`symmetric` *nil*))
Each feature is declared by binding it to `t` (the symbol for true), so
(`define-primitive-role` a_transitive_role `:transitive` *t*) defines a transitive role.
As opposed to ORM, you explicitly have to define a transitive role in RACER.

Important notice: currently reflexive roles are only permitted for $\mathcal{ALCH}$ knowledge bases, so they cannot be used since the semantics of lexons already impose a minimum of $\mathcal{ALCI}$.

[2]ORM is more expressive in this case: it also supports irreflexivity, asymmetry, antisymmetry, intransitivity and acyclicity of roles.

Frequency or Cardinality constraints

ORM frequency constraints on roles (generally called cardinality constraints) have a straightforward translation to qualified number restrictions in description logic.

Frequency constraints on concepts can be supported by applying the following rule: the inverse role of every mandatory role played by the restricted concept has to be constrained by the original object frequency constraint. The $Concept\_m$ in Table 4.2 denotes all concepts connected to the constrained $Concept_1$ via an inverse mandatory role.
This tweak is not a direct translation and requires some preprocessing to detect all mandatory roles. However, it captures the possible inconsistencies of conflicting cardinality constraints on roles connected to the cardinality-constrained concept.
(In case reflexive roles were applicable, we could have used a 'virtual' cardinality-constrained reflexive role on the restricted concept.)

Uniqueness constraints

Only 1:n and n:1 uniqueness constraints can be mapped to RACER. Naturally their equivalents are expressed as qualified number restrictions.

Many-to-many m:n uniqueness constraints are irrelevant for description logic reasoning since in mathematic sets there can be no duplicates. Such constraints only become important when a relational database is connected to a set of lexons, so we suggest to store them inside the bridging part of the commitment.

Similarly, external uniqueness constraints on roles find no translation and have to be ignored inside an ontological commitment. We have tried to find a way out of this by means of the available DL constructors, alas to no avail. Because of this, there are some potential inconsistencies that will be overlooked although this is quite unlikely to happen if we look at practical cases.

ORM uniqueness constraints are added to the conceptual schema only to implement complete reference schemes for a relational database mapping. Thus primary uniqueness constraints are also ignored during the verification phase of an ontological commitment.

Role subsets

Role subset constraints have to be translated by defining a subsumption between the concepts playing the respective roles.

It would be tempting to apply the role hierarchy constructor `implies` to express a role pair subset constraint, but since there is no unique role name assumption, this could cause semantic chaos. Therefore, a role pair subset constraint is translated as two single role subset constraints.

Role equality

Role equality constraints are simply stated by defining an equivalence between the concepts that are each involved in one of the two roles.

Role exclusions

Role pair exclusion for two roles R_1 and R_2 could be supported if it were legal to write (`implies` R_1 (*not* R_2)) and (`implies` R_1 (*not* R_2)). Unfortunately negation is only allowed on primitive concepts.

We have found a way to include role exclusion by stating that the concepts connected by the participating roles are disjoint. Because it is prohibited in RACER to assert disjointness between concept-expressions ((`disjoint` (`all` R_1 C_x)(`all` R_2 C_y))), we have to assign these two concept-expressions to two new concept names. For simplicity we have chosen to concatenate each concept name with its role name, as you can see in Table 4.2. Of course this method can be applied to any n-ary role exclusion constraint.

Value Constraints with RACER Concrete domains

As stated before, RACER allows a restricted value domain to be defined for its concepts, meaning instances can be of type integer or rational and there is a set of arithmetic operators supported to perform calculations, conversions, etc. Table 4.3 shows the complete BNF for the concrete rational number and integer domains.

$$\begin{array}{rll}
CDC \rightarrow & (\texttt{a}\ AN) & | \\
 & (\texttt{an}\ AN) & | \\
 & (\texttt{no}\ AN) & | \\
 & (\texttt{min}\ AN\ integer) & | \\
 & (\texttt{max}\ AN\ integer) & | \\
 & (\texttt{equal}\ AN\ integer) & | \\
 & (\texttt{>}\ aexpr\ aexpr) & | \\
 & (\texttt{>=}\ aexpr\ aexpr) & | \\
 & (\texttt{<}\ aexpr\ aexpr) & | \\
 & (\texttt{<=}\ aexpr\ aexpr) & | \\
 & (\texttt{=}\ aexpr\ aexpr) & \\
aexpr \rightarrow & AN & | \\
 & real & | \\
 & (\texttt{+}\ aexpr1\ aexpr1^*) & | \\
 & aexpr1 & \\
aexpr1 \rightarrow & real & | \\
 & AN & | \\
 & (\texttt{*}\ real\ AN) &
\end{array}$$

Table 4.3: RACER Concrete Domain Constructs

Although the name `real` is used, it refers to a rational number. The underlying reasoning scheme in RACER is based on $\mathbb{Q}$ and $\mathbb{Z}$ in combination with linear inequations.

Value constraints can be added to individuals or concepts by connecting them to so-called attributes, which have to be declared by their attribute name (AN) in the signature of the T-Box. By constraining the attribute connected to a certain concept, we can impose the original value constraint.

Apart from integer and rational number domains, RACER does not support general value constraints on concepts as in ORM, defining a restricted set or range of values e.g. by defining a set of string codes. There are two ways of sidestepping this problem:

1. The first way is to transform the domain restricted concept and its connecting role into a RACER feature. A feature is a special kind of role that has a restricted set of values as its domain is a value range instead of a (set of) instances of concept(s).
2. The second option is to build a hierarchy of concepts consisting of the value concept as top node and all possible values as disjoint subtypes. Of course the semantics of the range values get lost by applying this transformation.

However, we still choose to designate a RACER attribute to the restricted concept, and then constrain it according to the originally supplied value ranges. This method is preferred especially since it has been announced that future implementations of RACER will also support string concrete domains, which are commonly used (e.g. to define a set of codes like for the sexes 'F' and 'M').

Path expressions

Every Omega-RIDL path expression defines a set of instances inside a model in terms of the referenced concepts, possibly satisfying some constraints. In RACER an equivalent can be constructed, although the syntax is less apparent. A conceptual path that contains subsumptions is expressed through silent subtyping[3], meaning that only the subtype of a subsumption is mentioned, and not its supertype(s). The next example should make this clear:

Person drives **at-most-one** Porsche subsumed_by Car belonging_to Company

is equivalent in DL to

(`implies` Person (`at-most` 1 drives (`and` Porsche (`some` belonging_to Company))))

Car is omitted because the Porsche concept inherits all roles of its supertype. The subsumption is explicitly shown in Omega-RIDL for clarity. Note that there is no translation for this in description logic.

Path equivalence

To express the equivalence of two conceptual path expressions in $\mathcal{SHIQ}$, we simply have to define them to be equivalent. Path equivalence is a very useful feature that is rarely to be found in other languages.

An example: to state that a musician who plays in a group that plays certain genres plays all the genres that group plays and vice versa we could write this down in Omega-RIDL as Musician plays_in Group playing Genre `is-equivalent-to` Musician plays Genre .

[3]Silent subtyping was also a key feature of old RIDL implementations.

Conditional Statements

In general, conditional statements such as the one in Example 2 of Section 2.3.3 cannot be defined in description logic. There is however one kind of omega-RIDL if-then statement that can be translated, namely if-then-statements with condition and goal having identical concept names at the start[4] and containing no A-Box assertions.
A semantic equivalent is reached by defining a general inclusion axiom between the two path expressions inside the if- and the then-clause, so we can leave out conditionals statements from the consistency checking.

Unsupported Omega-RIDL Constraints

We now summarize all constraints that cannot be translated to the RACER $\mathcal{SHOIQ}(D)$ description logic.

- M:n uniqueness constraints on roles are only relevant for database purposes. We suggest to place them inside the bridging commitment part of a commitment.
- External uniqueness constraints are also mainly added for database purposes and cannot be verified together as the other T-Box constraints
- Most ORM ring constraints cannot be translated except for transitive and symmetric roles.
- Although currently not investigated, state transition constraints, which describe temporal changes of concepts are clearly not translatable into $\mathcal{SHIQ}$. They must be handled through another formalism, e.g. situation calculus.
- Just like the old RIDL, Omega-RIDL will probably contain procedural statements, with nested conditionals and combinations of A-Box and T-Box statements, etc.
  In general these will also not be translatable.

Note that this list still can grow in the future since the full syntax of Omega-RIDL was not available at the time of writing this thesis.

Another thing to mention is that we do not present a 'smart' method of translating, meaning that subsuming constraints are not detected and are translated on the spot. It could be interesting to add, but for now we only consider the mapping itself.

[4]We could loosen this precondition by allowing identical concepts at beginning or end of the if-condition and then-body, thereby reversing a conceptual path if the identical concepts are not at the front of both parts.

4.3.4 Inconsistency Analysis

RACER offers a large range of decision functions to answer the different consistency problems. We now present a selected overview of the most relevant functions for both A-Box and T-Box.

T-Box consistency

The following two functions are the first to be called from the Omega-RIDL compiler:

- (`check-tbox-coherence`) returns a list of all unsatisfiable concepts. If only a list containing `nil` is returned (`nil`), then this means that there is no synonym for the incoherent bottom concept and that the T-Box is consistent.
- (`concept-coherent?`) returns `nil` if there is an unsatisfiable concept, otherwise it returns `t` (true).
- (`classify-tbox`) classifies the T-Box and needs to be executed before queries can be posed.

These are some of RACER's subroutine functions:

- (`concept-satisfiable?` *concept-exp*1) returns `t` if *concept-exp*1 is satisfiable, `nil` otherwise.
- (`concept-subsumes?` *concept-exp*1 *concept-exp*2) returns `t` if *concept-exp*1 subsumes *concept-exp*2, `nil` otherwise.

To avoid redundancy: the outcome of the following self-evident functions is analogous to the ones above.

- (`concept-equivalent?` *concept-exp*1 *concept-exp*2)
- (`concept-disjoint?` *concept-exp*1 *concept-exp*2)
- (`concept-equivalent?` *concept-exp*1 *concept-exp*2)

A-Box consistency

The A-Box component of RACER will only come of real use in the second phase of the development of Omega-RIDL, when 'bridging commitments' (see Section 2.4.1) will be introduced in the DOGMA system, admitting instance assertions inside a commitment.
It could also be used for instance validation and for querying.
These are the most important A-Box functions:

- (`realize-abox`) checks the consistency of the A-Box and computes the most specific concepts for each individual in the A-Box.
- (`check-abox-coherence`) checks if the A-Box is consistent; returns `t` if consistent, otherwise information about the infringing instances is printed
- (`abox-consistent?`) returns `t` if the A-Box is consistent and `nil` otherwise

4.3.5 A Sample Translation

We shall now provide an example to demonstrate some of the mapping solutions explained above. The chosen ontology is situated in the field of music catalogues.

context	**termlabel1**	**role**	**co-role**	**termlabel2**
music_catalogue	Musician	plays_on	features	Track
music_catalogue	Musician	has_written	written_by	Track
music_catalogue	Musician	is_member_of	has_member	Band
music_catalogue	Musician	plays	played_by	Genre
music_catalogue	Band	is_influenced_by	has_influenced	Band
music_catalogue	Band	has	of	Url
music_catalogue	Band	made	made_by	Record
music_catalogue	Band	has	of	Name
music_catalogue	Musician	is_a	subsumes	Person
music_catalogue	Producer	is_a	subsumes	Person
music_catalogue	Person	has	of	Name
music_catalogue	Producer	produced	produced_by	Record
music_catalogue	Engineer	is_a	subsumes	Person
music_catalogue	Engineer	mixed	mixed_by	Track
music_catalogue	Track	is_on	contains	Record
music_catalogue	Track	has	of	Title
music_catalogue	Record	has	of	Title
music_catalogue	Record	has	of	Genre
music_catalogue	Record	released_on	released	Recordlabel
music_catalogue	Record	has	of	Releaseyear
music_catalogue	Record	has	of	Rating

Table 4.4: A music catalogue ontology - DOGMA lexons

The elimination of the context labels is straightforward in this case: there is only one context term present in front of the entire commitment, so it can simply be omitted.

```
\\ * ** preliminary Omega-RIDL commitment * * *
music_catalogue.{
Musician is-a person
Producer is-a person
Engineer is-a person

\\ all referenced lexons including co-roles
Musician plays_on [features] Track
Musician has_written [written_by] Track
Musician is_member_of [has_member] Band
...

Person has one Name
Band has one Name
Band has_member at-least 1 Musican
Musician plays_on at-least 1 Track
Track is_mixed_by at-least 1 Engineer
Track has one Title
Record has one Title
Record has at-least 1 Genre
Record released_on at-least 1 Recordlabel
Record produced_by at-least 1 Producer
Record contains at-least 2 Track
Record has one Releaseyear
Record has at-most 1 Rating
Record made_by at-least 1 Band

Person subsumes Producer excludes Person subsumes Musician
Person subsumes Musician excludes Person subsumes Engineer
Rating is-restricted-to {0..10}
is_influenced_by is-transitive

Musician is_member_of Band made Record has Genre
is-equivalent-to Musician plays Genre

}
```

Table 4.5: A music catalogue ontology - Omega-RIDL commitment

Figure 4.2 represents the ORM diagram of the ontological commitment written in Omega-RIDL as formulated in Table 4.5. Notice that it only shows the graphically displayable part of the commitment and that it already becomes quite complicated even though the ontology is still relatively small. Also, for this example, m:n uniqueness and external uniqueness constraints have not been supplied since they cannot be translated anyway.

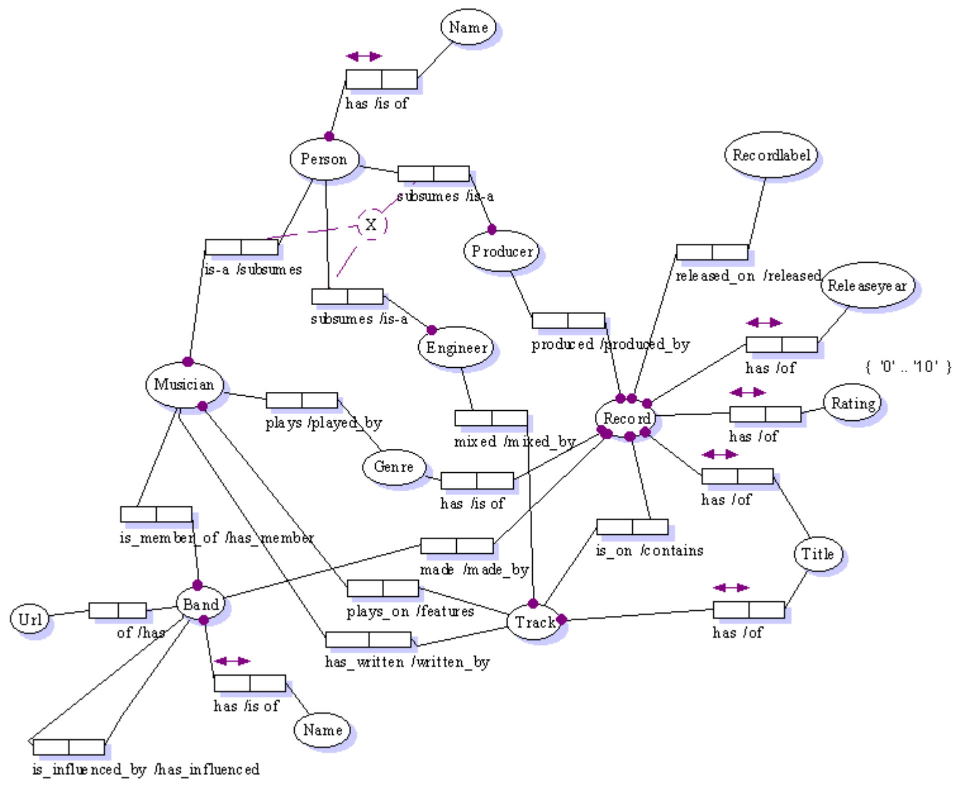


Figure 4.2: A music catalogue ontology - binary ORM diagram

The RACER translation of the above commitment is presented in Table 4.6. We have supplied domain and range constraints for some roles, which is not really necessary, but it improves RACER's reasoning and classification performance. This could be automated by checking for each role if it is exclusively defined in one lexon inside the commitment.

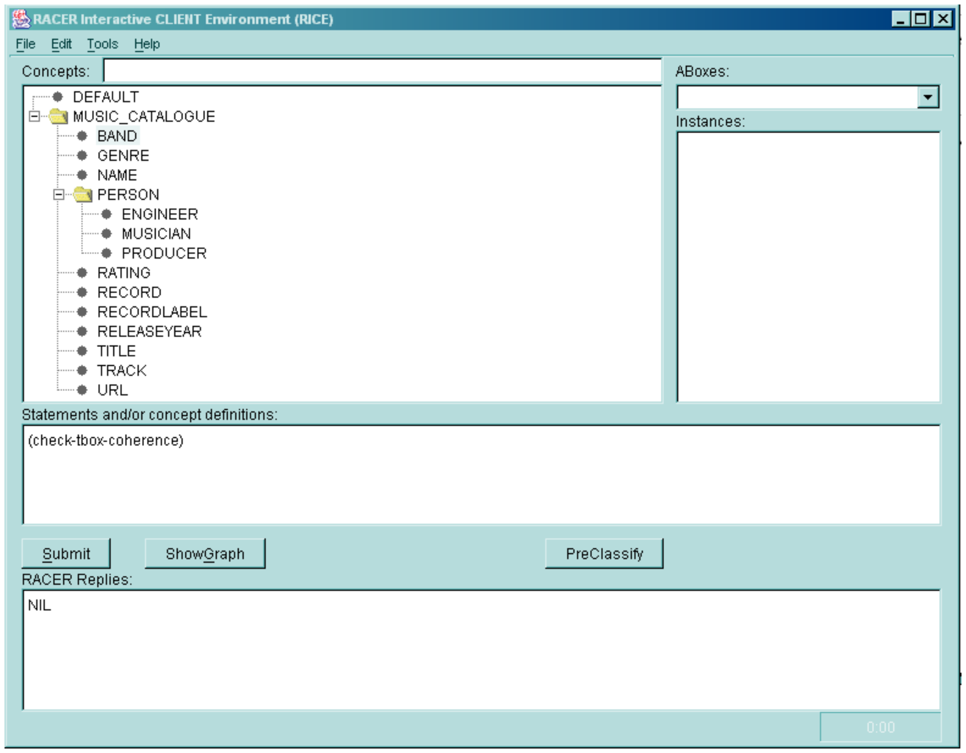
The chosen ontology proves to be satisfiable after submitting it to the RACER engine.

```
;;; initialize the T-Box "music_catalogue"
(in-tbox music_catalogue)

;;; the signature of the T-Box
(signature
:atomic-concepts (Person Musician Engineer Band Producer Url Record
                  Track Title Genre Recordlabel Releaseyear Rating)
:roles ((plays :inverse played_by)
        (plays_on :inverse features)
        (has_written :inverse written_by)
        (is_member_of :inverse has_member)
        (is_influenced_by :inverse influenced :transitive t)
        (mixed :inverse mixed_by)
        (produced :inverse produced_by)
        (has :inverse of)
        (released_on :inverse released)
        (made :inverse made_by)
        (contains :inverse is_on))
:attributes ((integer score)))

;;; domain & range restrictions for roles (enhances performance)
(domain produces Producer)
(range produces Record)
(domain released_on Record)
(range released_on Recordlabel)

;;; the (constrained) concepts
(implies Musician Person)
(implies Producer Person)
(implies Engineer Person)

(implies Person (exactly 1 has Name))
(implies Band (exactly 1 has Name))
(implies Band (at-least 1 has_member Musician))
(implies Musician (at-least 1 plays_on Track))
(implies Track (at-least 1 mixed_by Engineer))
(implies Track (exactly 1 has Title))
(implies Record (exactly 1 has Title))
(implies Record (at-least 1 has genre))
(implies Record (at-least 1 released_on Recordlabel))
(implies Record (at-least 1 produced_by Producer))
(implies Record (at-least 2 contains Track))
(implies Record (exactly 1 has Releaseyear))
(implies Record (at-most 1 has Rating))
(implies Record (at-least 1 made_by Band))

(equivalent Rating (and (min score 0) (max score 10)))

(implies (and Musician (some is_member_of (and Band
        (some made (and Record (some has Genre))))))
  (and Musician (some plays Genre)))
```

Table 4.6: A music catalogue ontology - RACER T-Box terminology

4.3.6 RICE: Graphical T- and A-Box Viewing

To facilitate the maintenance and of RACER knowledge bases, a graphical interface has been implemented. The system is called RICE (RACER Interactive Client Environment) [28] and provides a set of windowed views on a RACER Terminology. One can browse and inspect the T-Box and A-Box through a set of menus that provide the functionality of the JRACER API.



```
DEFAULT
MUSIC_CATALOGUE
    BAND
    GENRE
    NAME
    PERSON
        ENGINEER
        MUSICIAN
        PRODUCER
    RATING
    RECORD
    RECORDLABEL
    RELEASEYEAR
    TITLE
    TRACK
    URL
```

```
(check-tbox-coherence)
```

```
NIL
```

Figure 4.3: A music catalogue ontology - RICE screenshot (T-Box view)

A screenshot of RICE containing the T-Box of the translated music catalogue ontology is presented in Figure 4.3. As you can see there are no inconsistencies present since RACER does not show any insatisfiable concepts after querying for the T-Box coherence.

To give an inconsistent example, we consider a somewhat unrealistic change to the above commitment: suppose that every recordlabel released at least twenty records (a mandatory role), that every record is released on at least four recordlabels (cardinality constraint) and that there are only three recordlabels left in the world (object frequency constraint). This will add the following statements to the Omega-RIDL commitment (Table 4.7) and the RACER translation (Table 4.8):

```
\\ * ** preliminary Omega-RIDL commitment * * *
...
Recordlabel released at-least 20 Record
Record released_on at-least 4 Recordlabel
Record released_on at-least 3 Recordlabel
}
```

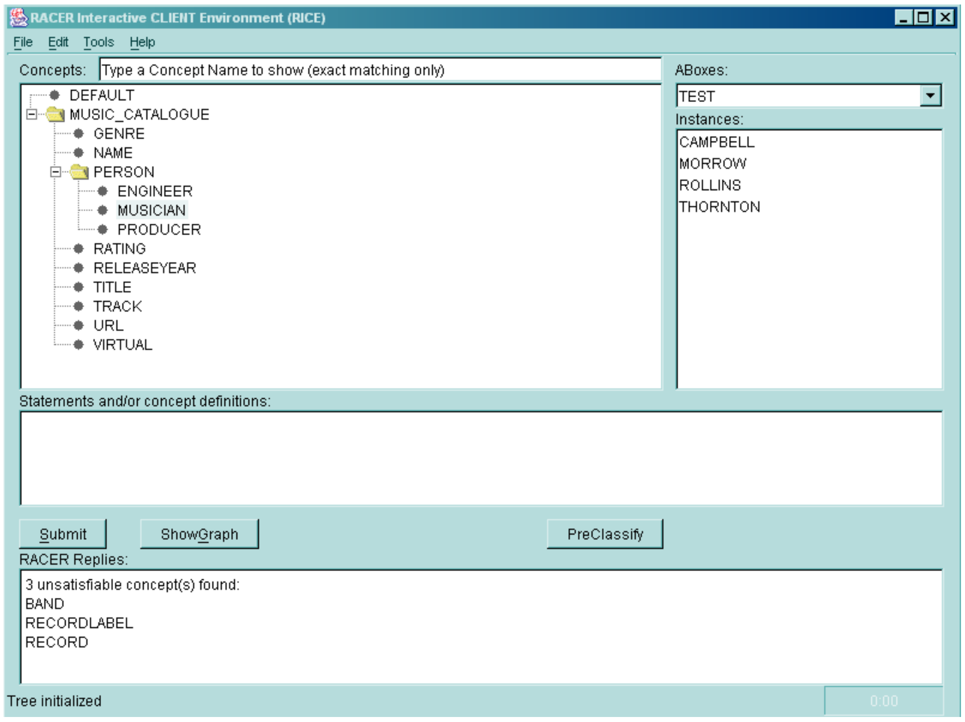
Table 4.7: A music catalogue ontology (extension) - Omega-RIDL commitment

```
...
(implies Recordlabel (at-least 20 released Record))
(implies Record (at-least 4 released_on Recordlabel))
(implies Record (at-least 3 released_on Recordlabel))
```

Table 4.8: A music catalogue ontology (extension) - RACER T-Box terminology

The inverse of the mandatory *released* role is constrained according to the rule in Table 4.2 and clearly conflicts with the other cardinality constraint on this same role.

After querying for the T-Box coherence, RICE gets the message from RACER that three concepts have become unsatisfiable as you can see in Figure 4.4. They are left out from the concept tree in the left-hand column.



```
3 unsatisfiable concept(s) found:
BAND
RECORDLABEL
RECORD
```

Figure 4.4: A music catalogue ontology (extension) - RICE screenshot

4.3.7 Decidability and Performance

Normally reasoning is undecidable for a general description logic, but $\mathcal{SHIQ}$ reasoning has been proven to be decidable [32].

Due to the heavy strain of inverse roles, the worst time complexity of the subsumption problem in $\mathcal{SHIQ}$ turned out to be NExpTime. Fortunately, by including a large number optimizations performance can be improved in many cases. We have not encountered any work on the behavior of $\mathcal{SIQ}$, but we expect that in practice the absence of role hierarchies will not make much difference.

Also, most inference engines, including RACER, are implemented in Lisp, so there will be some performance drop by connecting a Java API to a Lisp binary compared to working completely in Lisp, which can turn out to be considerably faster in a considerable number of cases [46].
Also not unimportantly for DOGMA, RACER has been proven to perform well on large knowledge bases [26].

4.4 Supporting A-Box Conditionals: Rule Engines Revisited

In business application domains, so-called business rules are very common. Business rules in their actual sense express organization policies, workflow, business processes, ...
Among these we find many constraining if-then-rules that have an A-Box query inside the conditional, e.g. "**if** Person has Car='red' **then** Person has_received **at-least** 1 SpeedTicket". RACER however does not support A-Box-conditioned T-Box constraints because it clearly separates instance from terminological level.

One solution to verify the consistency of these type of rules is to temporarily extend the current T-Box with every alternative constraint and then check the consistency.
In case of if-then-else clauses, the alternative ('else') case can be expressed as the conclusion of the negated condition. In general, negation is only allowed on atomic concepts in logic programming, but by applying De Morgan's laws, one can always convert to the negation normal form.

A better solution would be to include, next to an inference engine, a rule engine. This has been explored in the past, with good results. For example, SweetJess [20] is a tool that employs the JESS rule engine to inference RuleML [51] rules imposed on a DAML+OIL ontology.
Thus we could consider integrating the JESS engine into DOGMA.

In practice, this would mean an ontological commitment is validated through RACER, and the 'business' if-then-rules (that are left out from RACER) are then translated and inserted into JESS, together with the basic terminology (subsumptions between concepts). JESS supports Java calls from within its system, so business rules can have business actions (which can include new

fact assertions, T-Box constraints or external Java function calls) in their body, e.g. "**if** *John is_a Loyal_customer* **then** (($John\ has\ Discount = 5\%$) **and** ($mailsys.sendMail(John@yahoo.com, discountmsg)$)))".
Even more interesting, we could call the JRACER API directly from within JESS to directly add T-Box constraints when a dedicated rule is fired.

As JESS supports disjunctions, conjunctions and negations, there is much expressive potential and this could be a nice extra feature to the DOGMA system.

4.5 Incremental Consistency Checking

By only checking the consistency at its final design stage, an ontological commitment is fed as a whole into RACER, maximizing the debugging overhead.

To preserve a better overview of the consistent part of a commitment that is under construction, we could alternatively implement an interpreter that verifies the consistency step-by-step. In this manner, one can rapidly detect the first erroneous constraint instead of having to deal with multiple inconsistencies at the same time. The downside of this will probably be a remarkable decrease in performance, but this has to be verified.
Of course it will still be possible that adding one constraint leads to a cascade of conflicts, leaving the user with an untransparent set of inconsistencies.

The problem of pointing out the sources of inconsistency in a deterministic way is rather complex and is not a standard feature of an inference engine. An example implementation of it is available for the CLASSIC system [29].

4.6 Future Work

Once an applicable part of the Omega-RIDL syntax is finished, the proposed translation mapping to RACER input can be implemented. This mapping can then be used to check the consistency of commitments that are parsed by an Omega-RIDL interpreter.
In addition we could add the JESS rule engine to DOGMA to extend the system with support for business rules, if required.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis we have described the current status of the DOGMA system, introducing the Omega-RIDL constraint language to formally describe ontological commitments. We have shown that it is possible to translate Omega-RIDL constraints (which are backward compatible to binary ORM constraints) into description logic as these are largely overlapping subsets of first order logic. The close relationship between Omega-RIDL ontological commitments and knowledge representation terminologies in the $\mathcal{SHIQ}$ description logic enabled us to explore the functionalities of inference engines.

Furthermore we have demonstrated that the RACER inference engine comprises a useful addition to DOGMA to verify the consistency of ontological commitments. This has led to the concrete translation mapping of Omega-RIDL/binary ORM commitment syntax into $\mathcal{SHOIQ}(\mathcal{D})$ RACER input. Distinguishing between the T-Box and the A-Box, RACER provides a formal platform to check, validate and query commitments, implementing decidable consistency verification algorithms. Together with its technical specifications and add-ons, this makes it a perfect candidate to integrate into the DOGMA framework.
In addition we have concluded that a rule engine such as JESS could also be of use as a secondary inferencing tool to execute business rules.

5.2 Future work

A lot of related work for the DOGMA system is still in the pipeline.

One of the first tasks is to build an Omega-RIDL interpreter that employs the RACER engine as back-end system to verify the consistency of ontological commitments.
As an extension, an Omega-RIDL compiler can be implemented to store commitments in some output format. This will most probably be some XML format, presumably called Omega-RIDL-ML, which should be backward compatible to ORM-ML.

Another useful addition could be a graphical browsing utility for Omega-RIDL commitments, which could be plugged in to the DOGMA Modeler tool. To achieve this, we could the RICE client application for RACER to automatically generate a graphical presentation from a textual commitment, to get a better overview of the respective lexon terminology. Of course the ORM constraints will be lost then presentation-wise, so maybe a proper implementation will be preferred.

Also interesting to do is performing instance validation of a commitment by creating and querying an accompanying A-Box. This could be accomplished by employing the RACER A-Box query component. Recall that Omega-RIDL query path expressions are easy to translate as we have shown.

Since we have presented a concrete translation from Omega-RIDL and binary ORM to $\mathcal{SHIQ}$, our proposed mapping can be used as a guideline to define the formal translation of both formalisms into other ontology and rule languages such as DAML+OIL, RDF(S), RuleML, etc. and vice versa. This would be a major improvement because then many existing, external ontologies could be stored in the DOGMA format, which is ideal for testing and experimenting.

We finally emphasize that the true power of Omega-RIDL will emerge once there is a concrete to formulate bridging commitments (or lexical mappings), promoting varied ontology$\Leftrightarrow$database mapping functions to first-class procedures of a very high-level ontological commitment and query language.

Bibliography

[1] T. Halpin, *"Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design"*, Morgan Kaufmann Publishers, 2001.

[2] O. De Troyer, R. Meersman, F. Ponsaert, *"RIDL User Guide"*, (preliminary version), Control Data Corp., 1984.

[3] R. Meersman, F. Van Assche, *"Modelling and Manipulating Production Data Bases in Terms of Semantic Nets"*, IJCAI Karlsruhe, pp. 325-329, 1983.

[4] O. De Troyer, R. Meersman, P. Verlinden, *"RIDL* on the CRIS case: A Workbench for NIAM"*, INFOLAB Tilburg University, 1988.

[5] O. De Troyer, *"RIDL*: A Tool for the Computer-Assisted Engineering of Large Databases in the Presence of Integrity Constraints"*, INFOLAB Tilburg University, 1989.

[6] R. Meersman, *"The High-Level End User"*, Control Data Corp., 1982.

[7] G. Verheijen, J. van Bekkum, *"NIAM: aN Information Analysis Method"*, In Proceedings of IFIP Conference on Comparative Review of Information Systems Methodologies (Eds. Verrijn- Stuart, Olle, Sol). North Holland, 1982.

[8] A.C. Bloesch, T. Halpin, *"ConQuer: a Conceptual Query Language"*, In Proceedings of ER96: 15th International Conference on Conceptual Modeling, Springer LNCS, no. 1157, pp. 121-33, 1996.

[9] STARLab VUB, The DOGMA project, 2003
http://www.starlab.vub.ac.be/research/dogma.htm

[10] J. Demey, *"Modeling DOGMA Ontologies and their Commitments"*, licenciaatsthesis, STARLab, Vrije Universiteit Brussel, 2002.

[11] J. Demey, M. Jarrar, R. Meersman, *"A Conceptual Markup Language that supports interoperability between Business Rule modeling systems"*, In Proceedings of CoopIS 02: the Tenth International Conference on Cooperative Information Systems, Lecture Notes in Computer Science, Springer-Verlag, 2002.

[12] J. De Bo, P. Spyns, *"Multilingual ontology engineering in a refined DOGMA framework"*, Technical Report STAR-2003-13, 2003.

[13] J.C. Giarratano, *"CLIPS User's Guide version 6.20"*, http://www.ghg.net/clips/download/documentation/, 2002.

[14] Jess: the Expert System Shell for the Java Platform
http://herzberg.ca.sandia.gov/jess/

[15] C.L. Forgy, *"Rete: a fast algorithm for the many pattern/many object pattern match problem"*, Artificial Intelligence Vol. 19, pp.17-37, 1982.

[16] Haley Rule Engines
http://www.haley.com

[17] Cerebra Server^{tm}
http://www.networkinference.com

[18] K. Baclawski, M. M. Kokar, R. Waldinger, P. A. Kogut, *"Consistency Checking of Semantic Web Ontologies"*, In Proceedings of the first International Semantic Web Conference 2002, Sardinia, Italia, 2002.

[19] E. Franconi, Gary Ng, *"The i·com Tool for Intelligent Conceptual Modelling"*, In Proceedings of 7th International Workshop on Knowledge Representation meets Databases (KRDB'00), Berlin, Germany, 2000.

[20] B.N. Grosof, M.D Gandhe, T.W. Finin, *"SweetJess: Translating Daml-RuleML to Jess"*, In Proceedings of International Workshop on Rule Markup Languages for Business Rules on the Semantic Web, held at First International Semantic Web Conference, 2002.

[21] J. Kopena, W.C. Regli, *"DAMLJessKB: A Tool for Reasoning with the Semantic Web"*, Drexel University Philadelphia, 2002.

[22] I. Horrocks, *"Using an Expressive Description Logic: FaCT or Fiction?"*, In Proceedings of the Sixth International Conference on the Principles of Knowledge Representation and Reasoning (KR-98), 1998.

[23] RACER: Renamed ABox and Concept Expression Reasoner, 2003
http://www.fh-wedel.de/∼mo/racer/

[24] V. Haarslev, R. Möller, *"Consistency Testing: The RACE Experience"*, In Proceedings TABLEAUX'2000, Springer-Verlag, 2000.

[25] V. Haarslev, R. Möller, *"RACER System Description"*, In Proceedings of the International Joint Conference on Automated Reasoning, IJCAR'2001, 2001.

[26] V. Haarslev, R. Möller, *"High Performance Reasoning with Very Large Knowledge Bases"*, In Proceedings of the International Workshop in Description Logics 2000 (DL2000), Aachen, Germany, 2000.

[27] V. Haarslev, R. Möller, *"RACER Users Guide and Reference Manual Version 1.7.6"*, 2002.

[28] R. Cornet, RICE (Racer Interactive Client Environment) - Download page
http://www.b1g-systems.com/ronald/rice/

[29] D.L. McGuinness, *"Explaining Reasoning in Description Logics"*, PH.D. Thesis, State University of New Jersey, 1996.

[30] D.L. McGuinness, A.T. Borgida, *"Explaining Subsumption in Description Logics"*, In Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI'95, pages 816–821, Montreal, Canada, 1995.

[31] F. Baader, *"Description Logic Terminology"*, In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors, The Description Logic Handbook: Theory, Implementation, and Applications, pages 485-495. Cambridge University Press, 2003.

[32] M. Buchheit, F.M. Donini, A. Schaerf, *"Decidable Reasoning in Terminological Knowledge Representation Systems"*, Research Report RR-93-10, Deutsches Forschungszentrum fur Kunstliche Intelligenz, Saarbrucken, Germany, 1993.

[33] I. Horrocks, U. Sattler, S. Tobies, *"Practical Reasoning for Expressive Description Logics"*, In Proceedings of LPAR'99, LNCS, Tbilisi, Georgia, 1999.

[34] I. Horrocks, *"Reasoning with Expressive Description Logics: Theory and Practice"*, In Proceedings of the 18th International Conference on Automated Deduction (CADE 2002), 2002.

[35] I. Horrocks, U. Sattler, S. Tobies, *"Reasoning with individuals for the description logic SHIQ"*, In Proceedings of the 17th International Conference on Automated Deduction (CADE-17), number 1831 in Lecture Notes In Artificial Intelligence, pages 482-496, Springer-Verlag, 2000.

[36] A.Y. Levy, M. Rousset, *"CARIN: A representation language combining Horn rules and description logics"*, In Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96), pages 323–327, 1996.

[37] F. Rossi, *"Constraint Logic Programming"*, In Proceedings of ERCIM/Compulog Net workshop on constraints, Springer, LNAI 1865, 2000.

[38] E. Gelle, R. Weigel, *"Interactive Configuration using Constraint Satisfaction Techniques"*, Second International Conference on Practical Application of Constraint Technology, PACT-96, London, UK, 1996.

[39] B.N. Grosof, I. Horrocks, *"Description Logic Programs: Combining Logic Programs with Description Logic"*, In Proceedings of the Twelfth International World Wide Web Conference (WWW 2003), 2003. (To appear)

[40] I. Horrocks, U. Sattler, *"Ontology Reasoning in the $\mathcal{SHOQ}$ Description Logic"*, In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, 2001.

[41] C. Lutz, *"Adding Numbers to the SHIQ Description Logic - First Results"*, In Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002), Morgan Kaufman, 2002.

[42] A. Artale, E. Franconi, *“A Survey of Temporal Extensions of Description Logics”*, In Annals of Mathematics and Artificial Intelligence, 30(1-4), 2001.

[43] G. Alsa, C. Baral, *“Reasoning in Description Logic Using Declarative Logic Programming”*, In Proceedings of AAAI, 2002.

[44] N. Guarino, M. Carrara, P. Giaretta, *“Formalizing Ontological Commitments”*, In Proceedings of AAAI, 1994.

[45] D. Miller, *“A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification”*, Journal of Logic and Computation, 1(4):497–536, 1991.

[46] E. Gat, *“Lisp as an Alternative to Java”*, Intelligence, pp. 21-24, 2000.

[47] M.E. Stickel, R.J. Waldinger, V.K. Chaudhri, *“A Guide to SNARK”*, 2000.

[48] A. Waterson, A. Preece, *“Verifying Ontological Commitment in Knowledge-Based Systems”*. In Knowledge-Based Systems, 12, 45-54, 1999.

[49] DAML+OIL language, March 2001,
http://www.daml.org/2001/03/daml+oil-index.html

[50] Resource Description Framework Schema
http://www.w3.org/TR/rdf-schema/

[51] Rule Markup Language
http://www.dfki.uni-kl.de/ruleml/

Appendix A

Description Logic: $\mathcal{SHIQ}$ Semantics

Construct	**Syntax**	**Semantics**
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Concept	C	$\{a \in \Delta^{\mathcal{I}} \vert\ a \in C^{\mathcal{I}}\}$
Intersection	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
Union	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Value restriction	$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$
Existential quantification	$\exists R$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}}\}$
Existential restriction	$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$
Number restriction	$(\leqslant n\ R)$	$\{a \in \Delta^{\mathcal{I}} \vert \vert \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\} \vert \leqslant n\}$
	$(\geqslant n\ R)$	$\{a \in \Delta^{\mathcal{I}} \vert \vert \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\} \vert \geqslant n\}$
	$(= n\ R)$	$\{a \in \Delta^{\mathcal{I}} \vert \vert \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \vert = n\}$
Quantified number restriction	$(\leqslant n\ R.C)$	$\{a \in \Delta^{\mathcal{I}} \vert \vert \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \vert \leqslant n\}$
	$(\geqslant n\ R.C)$	$\{a \in \Delta^{\mathcal{I}} \vert \vert \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \vert \geqslant n\}$
	$(= n\ R.C)$	$\{a \in \Delta^{\mathcal{I}} \vert \vert \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \vert = n\}$
Role hierarchy	$R_1 \subseteq R_2$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a,b) \in R_1^{\mathcal{I}} \rightarrow (a,b) \in R_2^{\mathcal{I}}\}$
Transitive roles	$R \subseteq R^+$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

Table A.1: Semantics of $\mathcal{SHIQ}$ concept & role constructors

An interpretation $\mathcal{I}$ consists of a non-empty set $\Delta^{\mathcal{I}}$ (the domain of the interpretation) and an interpretation function, which assigns to every atomic concept C a set $C^{\mathcal{I}}$ and to every atomic role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Appendix B

Translation of Binary ORM into $\mathcal{SIQ}$

Binary ORM	**Abstract $\mathcal{SIQ}$ Syntax**
DOGMA.Lexon$\{Concept_1\ role\ [\ co\text{-}role\ ]\ Concept_2\}$	$Concept_1$, $Concept_2$, $role$ $co\text{-}role \equiv role^-$
DOGMA.Lexon$\{Concept_2\ is\text{-}a\ [subsumes]\ Concept_1\}$	$Concept_2 \sqsubseteq Concept_1$
ORM.InternalUniqueness$\{Concept_1\ role\ Concept_2\}$	$Concept_1 \sqsubseteq \leq 1\ role.Concept_2$
ORM.Mandatory$\{Concept_1\ role\ Concept_2\}$	$Concept_1 \sqsubseteq \geq 1\ role.Concept_2$
ORM.Mandat.IntUniqueness$\{Concept_1\ role\ Concept_2\}$	$Concept_1 \sqsubseteq = 1\ role.Concept_2$
ORM.Frequency$\{\leq n,\ Concept_1\ role\ Concept_2\}$	$Concept_1 \sqsubseteq \leq n\ role.Concept_2$
ORM.Frequency$\{\geq n,\ Concept_1\ role\ Concept_2\}$	$Concept_1 \sqsubseteq \geq n\ role.Concept_2$
ORM.Frequency$\{= n,\ Concept_1\ role\ Concept_2\}$	$Concept_1 \sqsubseteq = n\ role.Concept_2$
ORM.Subset$\{Concept_1\ role_1\ Concept_2\}$ $>\{Concept_1\ role_2\ Concept_3\}$	$Concept_1 \sqcap \exists role_1.concept_2$ $\sqsubseteq Concept_1 \sqcap \exists role_2.concept_3$
ORM.Equality$\{Concept_1\ role_1\ Concept_2\}$ $=\{Concept_1\ role_2\ Concept_3\}$	$(Concept_1 \sqcap \exists role_1.concept_2$ $\sqsubseteq Concept_1 \sqcap \exists role_2.concept_3)$ $\sqcap\ (Concept_1 \sqcap \exists role_2.concept_3$ $\sqsubseteq Concept_1 \sqcap \exists role_1.concept_2)$
ORM.Exclusion$\{Concept_1\ role_1\ Concept_2\}$ $\#\{Concept_1\ role_2\ Concept_3\}$	$\exists role_1.Concept_2$ $\sqsubseteq \neg\exists role_2.Concept_3$
ORM.Value$\{Concept_i,\ j..k\}$	$Concept_1 \sqsubseteq (\exists Concept_i.min_j \sqcap \exists Concept_i.max_k)$
ORM.Frequency$\{\leq \#n,\ Concept_1\}$	For all mandatory roles $role_m$ connecting $Concept_m$ to $Concept_1$: $Concept_m \sqsubseteq \leq \#n\ role_m^{-1}.Concept_1$
ORM.ExternalUniqueness$\{Concept_1\ role\ Concept_2,$ $Concept_3\ role\ Concept_2\}$	N/A
ORM.IntMNUniqueness$\{Concept_1\ role\ co\text{-}role\ Concept_2\}$	N/A

Table B.1: Binary ORM translated to $\mathcal{SIQ}$

The notation used in the first column is a formal notation for DOGMA lexons and ORM constraints which should be straightforward to apprehend.

The concrete $\mathcal{SIQ}$ syntax on the right provides a formal semantics for ORM constraints which has yet not been written down in any existing reference work to our knowing.
Role hierarchies (denoted by the $\mathcal{H}$ constructor symbol) are non-existent inside DOGMA: it is impossible to state e.g. that the role is_father is a subrole of the role is_parent, restricting its range.

Note that this is a mapping solely for binary ORM, which implies that n-ary roles and nested roles (also called objectifications) are not considered here.