DIP
Data, Information and Process Integration with Semantic Web Services
FP6 - 507483

Deliverable

D1.1
Report on the requirements analysis and the state of the art

Boris Motik
Stefano Spaccapietra

20 October 2006
EXECUTIVE SUMMARY

This deliverable overviews currently available logical frameworks for knowledge representation. The main focus is on description logic and F-Logic, since these two formalisms are currently considered as alternatives for ontology modeling in the Semantic Web. A detailed comparison of main features of both formalisms is included in this document. Based on a motivating use case, we note that none of the formalisms is sufficient for realizing complex applications. Rather, a hybrid framework, combining features of various formalisms is needed. Based on the feature comparison, a set of requirements on this framework are presented. In this way, a powerful framework for realizing complex Semantic Web applications is obtained.

The hybrid reasoning framework may influence the current standards for Semantic Web rule language. Furthermore, it will be realized by tools built in WP1.
### Document Information

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>FP6 – 507483</th>
<th>Acronym</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Full title</strong></td>
<td>Data, Information, and Process Integration with Semantic Web Services</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Document URL</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EU Project officer</strong></td>
<td>Daniele Rizzi</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliverable Number</th>
<th>1.1</th>
<th><strong>Title</strong></th>
<th>Report on the requirements analysis and state of the art</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work package Number</td>
<td>1</td>
<td><strong>Title</strong></td>
<td>Ontology reasoning and querying</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date of delivery</th>
<th>Contractual</th>
<th>M 6</th>
<th>Actual</th>
<th>2-Feb-04</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status</strong></td>
<td></td>
<td></td>
<td></td>
<td>version. 0.5 final</td>
</tr>
<tr>
<td><strong>Nature</strong></td>
<td></td>
<td></td>
<td></td>
<td>Prototype □ Report ✔ Dissemination □</td>
</tr>
<tr>
<td><strong>Dissemination Level</strong></td>
<td>Public ✔ Consortia □</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Authors (Partner)** Boris Motik (FZI)

**Responsible Author** Boris Motik  
**Email** [motik@fzi.de](mailto:motik@fzi.de)

**Partner** FZI  
**Phone** +49 721 9654812

**Abstract (for dissemination)** This document presents the state of the art in ontology reasoning and querying. In particular, it focuses on two formalisms considered nowadays for ontology reasoning and querying in the Semantics Web: description logics and F-Logic. A detailed comparative overview of features found in each of the formalisms is given. Based on this overview, a set of requirements for hybrid reasoning, to be realized in WP1, is derived.

**Keywords** description logic, F-Logic, reasoning, querying, hybrid reasoning

**Version Log**

<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>25-05-04</td>
<td>001</td>
<td>Boris Motik</td>
<td>Initial version created</td>
</tr>
</tbody>
</table>
## Project Consortium Information

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
</table>
| National University of Ireland Galway | NUIG | Prof. Dr. Christoph Bussler  
Digital Enterprise Research Institute (DERI)  
National University of Ireland, Galway  
Galway  
Ireland  
Email: chris.bussler@deri.ie  
Tel: +353 91 512460 |
| Fundacion De La Innovacion.Bankinter | Bankinter | Monica Martinez Montes  
Fundacion de la Innovation. Bankinter,  
Paseo Castellana, 29  
28046 Madrid,  
Spain  
Email: mmminez@bankinter.es  
Tel: 916234238 |
| Berlecon Research GmbH | Berlecon | Dr. Thorsten Wichmann  
Berlecon Research GmbH,  
Oranienburger Str. 32  
10117 Berlin,  
Germany  
Email: tw@berlecon.de  
Tel: +49 30 2852960 |
| British Telecommunications Plc. | BT | Dr John Davies  
BT Exact (Orion Floor 5 pp12)  
Adastral Park Martlesham,  
Ipswich IP5 3RE,  
United Kingdom  
Email: john.nj.davies@bt.com  
Tel: +44 1473 609583 |
| Swiss Federal Institute of Technology, Lausanne | EPFL | Prof. Karl Aberer  
Distributed Information Systems Laboratory  
École Polytechnique Fédérale de Lausanne  
Bât. PSE-A  
1015 Lausanne, Switzerland  
Email: Karl.Aberer@epfl.ch  
Tel: +41 21 693 4679 |
| Essex County Council | Essex | Mary Rowiatt,  
Essex County Council,  
PO Box 11, County Hall, Duke Street, Chelmsford, Essex, CM1 1LX,  
United Kingdom.  
Email: maryr@essexcc.gov.uk  
Tel: +44 (0)1245 436524 |
| Forschungszentrum Informatik | FZI | Andreas Abeckers  
Forschungszentrum Informatik  
Haid-und-Neu Strasse 10-14  
76131 Karlsruhe,  
Germany  
Email: aabcker@fzi.de  
Tel: +49 721 9654 0 |
<p>| Institut für Informatik, Leopold-Franzens | IFI | Prof. Dieter Fensel |</p>
<table>
<thead>
<tr>
<th>Universität Innsbruck</th>
<th>Institute of computer science University of Innsbruck Technikerstr. 25 A-6020 Innsbruck, Austria Email: <a href="mailto:dieter.fensel@uibk.ac.at">dieter.fensel@uibk.ac.at</a> Tel: +43 512 5076485</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILOG SA</td>
<td>Christian de Sainte Marie 9 Rue de Verdun, 94253 Gentilly, France Email: <a href="mailto:csma@ilog.fr">csma@ilog.fr</a> Tel: +33 1 49082981</td>
</tr>
<tr>
<td>inubit AG</td>
<td>Torsten Schmale, inubit AG Lützowstraße 105-106 D-10785 Berlin, Germany Email: <a href="mailto:ts@inubit.com">ts@inubit.com</a> Tel: +49 307261120</td>
</tr>
<tr>
<td>Intelligent Software Components, S.A.</td>
<td>Dr. V. Richard Benjamins, Director R&amp;D Intelligent Software Components, S.A. Pedro de Valdivia 10 28006 Madrid, Spain Email: <a href="mailto:rbenjamins@isoco.com">rbenjamins@isoco.com</a> Tel. +34 913 349 797</td>
</tr>
<tr>
<td>Net Dynamics Internet Technologies GmbH u. Co KG</td>
<td>Peter Smolle Net Dynamics Internet Technologies GmbH &amp;. Co KG Prinz-Eugen-Strasse 68-70 A-1040 Wien, Austria Email: <a href="mailto:peter.smolle@netdynamics-tech.com">peter.smolle@netdynamics-tech.com</a> Tel.: +43 1 503982615</td>
</tr>
<tr>
<td>The Open University</td>
<td>Dr. John Domingue Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, United Kingdom Email: <a href="mailto:j.b.domingue@open.ac.uk">j.b.domingue@open.ac.uk</a> Tel.: +44 1908 655014</td>
</tr>
<tr>
<td>SAP AG</td>
<td>Dr. Elmar Domer SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany Email: <a href="mailto:elmar.dorner@sap.com">elmar.dorner@sap.com</a> Tel: +49 721 6902 31</td>
</tr>
<tr>
<td>Sirma AI Ltd.</td>
<td>Atanas Kiryakov, Ontotext Lab, Sirma AI EAD, Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse, Sofia 1784, Bulgaria Email: <a href="mailto:atanas.kiryakov@sirma.bg">atanas.kiryakov@sirma.bg</a> Tel.: +359 2 9768 303</td>
</tr>
<tr>
<td>Tiscali Österreich GmbH</td>
<td>Tiscali Dieter Haacker</td>
</tr>
<tr>
<td>Company</td>
<td>Address</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>Tiscali Österreich GmbH.</td>
<td>Diefenbachgasse 35, A-1150 Vienna, Austria</td>
</tr>
<tr>
<td>Unicorn Solution Ltd.</td>
<td>Unicorn Solutions Ltd, Malcha Technology Park 1, Jerusalem 96951, Israel</td>
</tr>
<tr>
<td>Vrije Universiteit Brussel</td>
<td>Vrije Universiteit Brussel, Pleinlaan 2, G-10, 1050 Brussel, Belgium</td>
</tr>
</tbody>
</table>
**TABLE OF CONTENTS**

**EXECUTIVE SUMMARY** ............................................................................................................. I
**TABLE OF CONTENTS** ................................................................................................................ VI

1 **INTRODUCTION** ......................................................................................................................... 1

2 **MOTIVATING EXAMPLE** ............................................................................................................. 2

3 **STATE-OF-THE-ART IN ONTOLOGY REPRESENTATION** ...................................................... 5
   3.1 First-order Logic ......................................................................................................................... 5
   3.2 Description Logics ..................................................................................................................... 8
      3.2.1 Syntax .................................................................................................................................. 8
      3.2.2 Semantics ............................................................................................................................. 9
      3.2.3 Description Logics and OWL ............................................................................................... 10
      3.2.4 Example ............................................................................................................................. 11
   3.3 F-Logic ....................................................................................................................................... 15
      3.3.1 Objects and Classes in F-Logic ............................................................................................ 15
      3.3.2 Type System ....................................................................................................................... 17
      3.3.3 Inheritance ......................................................................................................................... 17
      3.3.4 Example ............................................................................................................................. 18
   3.4 Conceptual Modelling ............................................................................................................... 20
      3.4.1 Overview ............................................................................................................................. 20
      3.4.2 Example ............................................................................................................................. 21

4 **FEATURE COMPARISON** ......................................................................................................... 26
   4.1 Domains and Ranges ............................................................................................................... 26
   4.2 Closed vs. Open Worlds ......................................................................................................... 29
   4.3 Disjunctive Reasoning .............................................................................................................. 32
   4.4 Existential Quantification ...................................................................................................... 33
   4.5 Axioms ...................................................................................................................................... 35
   4.6 Decidability ............................................................................................................................... 36
      4.6.1 What is Decidability? ......................................................................................................... 36
      4.6.2 Practical Considerations ................................................................................................... 36
      4.6.3 Origins of Undecidability ................................................................................................. 37
   4.7 Logic-based vs. Conceptual Modelling Approach ................................................................... 40
      4.7.1 Data Modelling ................................................................................................................ 40
      4.7.2 Data Manipulation .......................................................................................................... 41
      4.7.3 Constraints ....................................................................................................................... 42
      4.7.4 Beyond Data Structures .................................................................................................. 42
4.7.5 Hybrid Paradigm ................................................................. 42

5 PRACTICAL REQUIREMENT: HYBRID REASONING .................... 44
  5.1 Supported Formalisms ......................................................... 44
  5.2 Level of Interoperability ..................................................... 44
  5.3 Decidability ................................................................. 45
  5.4 Open- and Closed-World Assumption ................................. 45
  5.5 Support for Non-monotonic Negation ................................. 45
  5.6 Support for Arbitrary Axioms ............................................. 45
  5.7 Acceptable Performance Level ............................................ 45
  5.8 Follow the Principle of ‘Graceful Degradation’ ....................... 46

6 CONCLUSION ........................................................................... 47

REFERENCES ............................................................................. 48
1 INTRODUCTION

The unprecedented success of the World Wide Web has inspired many visionaries with a vision of the New Economy, in which companies perform most of their business transactions online. The superiority of the new economy over the old one is based on the fact that via a global communication network, businesses all over the world can be interconnected, in order to interact directly with each other. In this way, intermediaries are eliminated from the business chain, thus improving the competitiveness of companies. Furthermore, direct communication between business partners provides opportunities for responding faster to business events and thus significantly optimizing the business processes.

The promises of the new economy have not yet been realized up to their full potential. Although the communication infrastructure currently has excess capacity, it is still not used in the way as envisaged by the new economy. The aspect that was often neglected by the visionaries is that this vision can be realized only if involved parties share some common understanding. Achieving this uniformity across companies and their IT infrastructure has shown itself to be a daunting task. This is mainly due to the immense amount of heterogeneity inherent in the systems. The main source of this heterogeneity is the difference in the conceptualization of the real world. Imagine, for an instance, a provider of an integrated portal for booking a holiday trip. The portal provider aggregates the data from hundreds of different providers in order to provide an integrated view. However, some providers may talk about ‘Hotels’, whereas others may talk about ‘Accommodation’. Direct aggregation of information is not possible, since different terminology is used.

Semantic Web Services are increasingly seen as the key technology to providing a solution to the challenges of the new economy. Web services and associated standards, such as SOAP, WSDL, RosettaNet or BPEL4WS, provide a communication interface and a workflow management protocol for business systems. However, these technologies do not address the conceptual heterogeneity inherent in the data that these systems exchange. Semantic technologies, such as ontologies and related formalisms are seen as a potential solution to this problem. Hence, the WP1 of the DIP project should extend existing standards for ontology modelling and reasoning in the Web with the features deemed necessary for applying semantic technologies to the mentioned integration problems.

The main goal of this document is to:

- present and overview the currently existing semantic technologies,
- point out the similarities and differences among them,
- create a list of requirements and problems that should be addressed by WP1.

To make the discussion concrete, in the next section we introduce a running real-world integration scenario. At the time of this writing, use case studies have not been available yet. Furthermore, to easily explain particular aspects of various logical formalisms more easily, we have decided to present a very simple example. Most use cases produced within the DIP project will be too complex for the purpose of an example.
2 MOTIVATING EXAMPLE

Let INT be a company that wants to provide an integrated tourism portal. Using the portal, customers can book a plane ticket, a hotel, a rental car or similar services. INT does not itself own or manage any of these resources. Instead, it simply integrates various offers by numerous large and small providers; we call these providers $S_i$. Figure 1 presents the structure of the portal.

INT needs to communicate potentially with hundreds of providers. Freedom to flexibly add or change providers it operates with is a main concern for INT. Currently, this is not an easy task, since, typically, each $S_i$ uses its own proprietary system to classify various entities it deals with.

We present next a concrete example of the differences in the conceptualization that INT may encounter. Let us assume that there are two providers of information about hotels, TUI and Thomas Cook, TC for short. Let us assume that both of them use different standards for representing information about hotels (this is not an unrealistic assumption at all).

Figure 2 presents the conceptualization that TUI uses to describe his offers. It is drawn using a simple notation. Oval nodes (e.g. Hotel) denote conceptual entities. Nodes without a surrounding oval (e.g. name) denote properties of the corresponding entities. Labelled arcs between oval nodes denote a relationship between the conceptual entities. Non-labelled arcs link a conceptual entity to its properties.

Figure 1. Description of the Integration Problem
On the other hand, TC uses a different conceptualization of his business domain. In particular, it does not deal exclusively with hotels, but provides a more fine-grained division into different types of accommodation, such as hotels, bungalows and apartments. Furthermore, information about hotels is structured differently. For example, geographical information is more fine-grained, as it represents the country region and the region within the city where a certain hotel is situated. Furthermore, TC also has the notion of a room type, but uses it to differentiate standard and deluxe rooms. Thus, the semantics of the notion of a room type at TC differs from the one at TUI, where it is used to characterize the number of beds in the room. The structure of rates is also different: for advertisement purposes, TC keeps always the standard rate and the reduced rate, so that the customers see how much they are able to save. Figure 3 shows the conceptualization as used by TC.

INT is now faced with a daunting task. In order to create its portal, it needs to bridge the semantic differences between the information coming from TUI and from TC. Customers of INT should be able to make a request for a hotel, possibly specifying all desired options, and obtain a list of hotels offered by either TUI and of TC. To do that, INT must create his own conceptualization of the tourism domain. It may either accept one
of the existing conceptualizations, choose some of the existing standards, or develop its own proprietary conceptualization. Whichever path it chooses, its main guiding principle should be to optimize for change. Namely, it is very likely that in near future, some of the providers will not be relevant, and that new providers will emerge. Hence, flexibility of the conceptualization is a core issue for INT. The conceptualization will be expressed in certain conceptual modelling formalism. Hence, it would be more precise to say that the flexibility of the formalism is a main issue. In the rest of this document we present several modelling formalisms which are currently under discussion in academic and industry circles.

After choosing the conceptualization formalism and creating its own conceptualization, INT has to describe how data in each source relates to the common conceptualization. Many conceptualization formalisms offer different primitives for performing this task. We focus in this document mainly on logical conceptualization formalisms, because their inference capabilities are commonly seen as key to solving this problem. For example, dependencies between the conceptualization of INT and those of the sources can be expressed as logical axioms, which, when executed, can be used to actually perform data integration. Hence, we shall present the available capabilities for representing axioms for formalisms considered.

Conceptual modelling approaches most frequently use data manipulation languages to express the mapping between a global integrated schema (in the example, the INT conceptualization) and its corresponding source schemas. Alternatives for the description of these mappings will be briefly discussed as well.

Finally, after correspondences have been established, means for querying the integrated conceptualization are required. Apart from expressiveness, an important feature of a query language is its computational complexity. Namely, choosing a more expressive query language is certainly going to increase the worst-case complexity of the language. Finding the right balance is almost an art. In the rest of this document, we shall overview the associated query languages for considered formalisms as well.

We finish this section with a note that recently, the term ‘ontology’ has established itself as denoting a conceptualization shared by some community. Hence, an ontology is somewhat stronger than a conceptualization, since it implies a consensus of a broader group of people. In the rest of this document we shall use the terms conceptualization and ontology interchangeably.
3 State-of-the-art in Ontology Representation

In this section we briefly overview some significant ontology representation approaches that we feel are particularly relevant for DIP. Although we shall try to cover in various formalisms in depth, we assume some elementary knowledge in mathematical logic (for a good introduction to the topic see [2]) and in data models. In the next section we conduct a comparative discussion, where we contrast different features of the formalisms considered. For each formal language, we discuss the following notions: epistemology, vocabulary, syntax, and semantics:

- At the epistemological level, the meta-model or the data-model is determined (coining representation means such as terms, variables, classes, predicates, roles, facets, etc.)
- The vocabulary (or signature), determines the dictionary and the alphabet of the language – what symbols can be used to denote what elements, etc. For example, there is often a requirement that the names of the functions and the predicates are mutually disjoint sets.
- The syntax determines what sort of expressions can be composed using the vocabulary.
- The semantics determines the meaning of the expressions of the language. This is the basis for any further automatic interpretation and reasoning.

3.1 First-order Logic

No overview of logical formalisms for ontology modelling would be complete without the mention of first-order logic, as it acts as the theoretical foundation of all other logics. An excellent introduction to first-order logic has been given in [2]. Here, we shall briefly recall its syntax and semantics, and discuss why it is so important for ontology modelling.

A first-order signature $\Sigma$ is a triple $(F, P, V)$, where $F$ is a finite set of function symbols, $P$ a finite set of predicate symbols, and $V$ a countable set of variables. Each function and predicate symbol has a non-negative integer arity; function symbols of zero arity are called constants, whereas predicate symbols of zero arity are called propositional variables. The set of terms $T$ over $\Sigma$ is defined as the smallest set such that (i) each variable is a term and (ii) if $f$ is a function symbol of arity $n$ and $t_1, ..., t_n$ are terms, then $f(t_1, ..., t_n)$ is a term. If $P$ is a predicate symbol of arity $n$ and $t_1, ..., t_n$ are terms, then $P(t_1, ..., t_n)$ is an atomic formula. The set of formulae $F$ over $\Sigma$ is defined as the smallest set such that: (i) each atomic formula is a formula, and (ii) if $\phi_1$ and $\phi_2$ are formulae and $x$ a variable, then $\neg\phi$, $\phi_1 \land \phi_2$, $\phi_1 \lor \phi_2$, $\exists x: \phi$ and $\forall x: \phi$ are formulae as well. Notation $\phi_1 \rightarrow \phi_2$ is often used as an abbreviation for $\neg\phi_1 \lor \phi_2$, and $\phi_1 \leftrightarrow \phi_2$ is often used as an abbreviation for $(\phi_1 \rightarrow \phi_2) \land (\phi_1 \leftarrow \phi_2)$.

Using first-order logic it is possible to express certain relations in the conceptualizations from Section 2. For example, it is possible to express that hotels are a type of accommodation:

$$\forall x: \text{(Hotel}(x) \rightarrow \text{Accommodation}(x))$$
In the expert circles, there is an endless ongoing discussion of what an ontology ‘means’. First of all, we advocate the philosophical view that meaning is something occurring exclusively in the minds of the humans who read the ontology. Since an ontology is meant to be a shared conceptualization of some domain, its goal is to evoke similar mental structures in the minds of the humans which read and use the ontology. An important step in this direction is achieved by using descriptive labels for ontology elements. In the above example, using the predicate symbols Hotel and Accommodation is crucial for intuitive understanding of the sentence, which basically says “if x is a hotel, then x is an accommodation”. For the computer, the following seemingly ‘meaningless’ sentence

$$\forall x: (\text{sym}102(x) \rightarrow \text{sym}1078(x))$$

has the same meaning as the previous seemingly ‘meaningful’ sentence. The computer cannot tell the difference, except for the fact that symbols used are different. The point of this discussion is the following:

- Using common labels in ontologies is crucial for common understanding.
- The semantics of labels is, however, an issue for humans. For the computer, the name Hotel has as much ‘semantics’ as the name sym102.
- Formal semantics of an ontology language defines the meaning from a computational point of view. The ‘semantic meaning’ given by formal semantics is quite different from the ‘semantic meaning’ given by labels; these two should not be confused.

We hope that by now it is clear that, while common labelling is important from the practical point of view, issues related to common labelling rapidly depart from the core competencies of the computer science, but require psychological and sociological competencies. These issues are not a core focus of WP1 of DIP, which has a more formal character. Hence, in the rest of this document we are concerned with the formal semantics primarily.

The formal semantics of first-order logic is defined by so called model theory. An interpretation is a pair $$(D, I)$$, where D is a countable non-empty domain set, and $I$ is a function which assigns to each n-ary predicate symbol P of S an interpretation $$P^I \subseteq D^n$$, and to each n-ary function symbol f a function $$f^I : D^n \rightarrow D$$. In this way, the interpretation function $$I$$ gives ‘meaning’ to symbols by associating some real relation or function with each relation or function symbol.

A variable assignment A is a function $$A : V \rightarrow D$$. An x-variant of A, written $$A_x$$, is an assignment which assigns the same values to all variables as A, except possibly to the variable x. For a given interpretation I and a variable assignment A, we defined a value of a term and formula in I and A:

- The value of a term t in I and A, written $$v^{I,A}(t)$$, is defined as follows:
  - For t a variable x, $$v^{I,A}(x) = A(x)$$.
  - Otherwise, for $$t = f(s_1, \ldots, s_n)$$, $$v^{I,A}(t) = f^I(v^{I,A}(s_1), \ldots, v^{I,A}(s_n))$$.
- The value of a formula $$\phi$$ in I and A, written $$v^{I,A}(\phi)$$, is defined as follows:
  - $$v^{I,A}(P(s_1, \ldots, s_n)) = \text{true}$$ if and only if $$(v^{I,A}(s_1), \ldots, v^{I,A}(s_n)) \in P^I$$. 
o $\mathcal{J}_{I,A}(-\phi) = \text{false}$ if and only if $\mathcal{J}_{I,A}(\phi)$ is true,

o $\mathcal{J}_{I,A}(\phi_1 \land \phi_2) = \text{true}$ if and only if both $\mathcal{J}_{I,A}(\phi_1)$ and $\mathcal{J}_{I,A}(\phi_2)$ are true,

o $\mathcal{J}_{I,A}(\phi_1 \lor \phi_2) = \text{true}$ if and only if some $\mathcal{J}_{I,A}(\phi_1)$ or $\mathcal{J}_{I,A}(\phi_2)$ is true,

o $\mathcal{J}_{I,A}(\exists x : \phi) = \text{true}$ if and only if there is a variable assignment $A_x$ which is an $x$-variant of $A$, such that $\mathcal{J}_{I,A_x}(\phi) = \text{true},$

o $\mathcal{J}_{I,A}(\forall x : \phi) = \text{true}$ if and only if for each $x$-variant $A_x$ of $A$, $\mathcal{J}_{I,A}(\phi) = \text{true}.$

A formula $\phi$ is true in an interpretation $I$ if $\mathcal{J}_{I,A}(\phi) = \text{true}$ for all variables assignments. In this case we say that $I$ is a model of $\phi$ and write $I \models \phi$. A formula $\phi$ is satisfiable if it has a model. Interpretation $I$ is a model of a set of formulae $S$, written $I \models S$, if $I \models \phi$ for each $\phi \in S$. $S$ entails $\phi$, written $S \models \phi$, if, for each $I$ such that $I \models S$, $I \models \phi$.

In first-order logic, the basic inference is satisfiability checking, that is, determining whether some formula has a model. This is because $S \models \phi$ if and only if $S \cup \{ -\phi \}$ is unsatisfiable.

Formal semantics allows us to formally reason about the ontological conceptualization of a world. In another words, it is possible to entail that some properties follow from a set of axioms. For example, let $S$ be the following set of first-order formulae:

$$\forall x: (\text{Hotel}(x) \rightarrow \text{Accommodation}(x))$$

$$\text{Hotel}(\text{savoy})$$

Now $S$ entails $\text{Accommodation}(\text{savoy})$, and here is why. Let $I$ be some model; then, it must satisfy both formulae. For the second formula to be satisfied, it must be that $\text{savoy}^I \in \text{Hotel}^I$. Furthermore, for the first formula to be satisfied, for all $x$, either $\neg \text{Hotel}(x)$ or $\text{Accommodation}(x)$ must hold. This must also hold for the valuation in which $x$ is assigned to $\text{savoy}^I$. Since $\text{savoy}^I \in \text{Hotel}^I$, the first part cannot be true, so the second part is true, i.e. $\text{savoy}^I \in \text{Accommodation}^I$. Hence, $S \models \text{Accommodation}(\text{savoy})$.

Although first-order logic is very expressive and comes with a very powerful inference mechanism, it is usually not directly used for ontology modelling. For one, first-order logic is a generic logical framework and is not particularly tailored for knowledge representation. A logic specially designed for knowledge representation will provide modelling primitives more suitable for modelling concepts, relationships and instances. This situation can be compared to C and C++ programming languages. Everything which can be done in C++, can be done in C. However, writing a program in C++ is usually easier, since one has modelling constructs tailored better to the problem being solved.

Another big disadvantage of first-order logic is its high computational complexity: it is a semi-decidable formalism. For a detailed discussion of decidability, please refer to Subsection 4.6.

Because of these concerns, we do not further elaborate first-order logic. However, as we shall see, first-order logic lies at the hart of most ontology modelling formalisms.

We finish this section with a note that there are numerous logics beyond the first-order logic, such as modal logic, three-valued logic and intuitionist logic, to name just a few. Since these logics are currently not considered for ontology modelling in the Semantic Web, we do not elaborate on them here, but refer the interested reader to [25] for a
comprehensive introduction. Furthermore, first-order logic has been extended with several non-monotonic formalisms, such as circumscription [26] and default logic [27], autoepistemic logic [28]. All of these non-monotonic logics attempt to formalize the notion of closed-world knowledge. Finally, by allowing variables to quantify over predicate names, we get second- and higher-order logics, also known as the type theory [29].

3.2 Description Logics

Description logics, also called terminological logics in the eighties, were initially developed as a reaction to the deficiencies of the frame-based knowledge representation systems which were popular in the mid-eighties. The first such system was KL-ONE [3]. It was characterized by a knowledge representation language with a clear model-theoretic semantics, which precisely governs the meaning of allowed inferences. For the difference, the semantics of inferences drawn by the frame-based knowledge representation systems of that time was given either informally or procedurally. Hence, KL-ONE made a significant step in the right direction.

After initial applications with KL-ONE, it became clear that in numerous applications decidability is an important feature. If a logic is undecidable, this means that users may ask queries that will never terminate. For real-world applications this is quite inconvenient, since an application which gets stuck in an endless loop every once in a while is of limited use.

Another reason to work with decidable logic is that a decidable logic will be much more amenable for optimizations than an undecidable logic. Namely, in a decidable logic one can isolate certain patterns of hard problems, which can then be optimized by some techniques. For an undecidable logic this is much more difficult. Because of these considerations, decidability is the prime feature of the description logics which are currently being considered for practical usage.

3.2.1 Syntax

In this section we present the SHOIQ(D) description logic, which is nowadays considered to be a description logic offering a good trade-off between expressivity and computational complexity. A description logic ontology consists of concepts, roles and individuals. Concepts represent sets of real-world entities, individuals are the real-world entities and can belong to some concept, and roles represent connections between individuals.

Apart from representing abstract notions, a practical ontology modelling language needs a way to model concrete data, such as numbers, strings, or intervals. To this purpose, so-called concrete domain is used, which provides a certain set of predicates with a predefined definition. For example, a predicate $<_{10}$ may be interpreted as the set of all integers smaller than 10.

Concepts in SHOIQ(D) can be atomic, which means they are denoted by their names. For example, to describe the conceptualizations of TUI from Section 2, we might use the atomic concepts Hotel, City and Rate. Apart from atomic concepts, one can use so-called defined concepts, which can have the syntactic structure as defined below, where $A$ denotes the atomic concept, $C$ and $D$ are defined concepts, $R$ is a role, $n$ is an integer and $d$ is a concrete predicate:
For example, the concept \( \text{Hotel \cap \exists located-in.EnglishCity} \) represents a hotel which is located in an English city. Similarly, \( \text{Hotel \cap \exists offers.(\forall amount.<_{100}}) \) is a hotel which offers a rate with a price smaller than 100 EUR.

A description logic ontology consists of a set of axioms specifying the properties about the concepts and properties. These axioms can be of the following form, where \( C \) and \( D \) are concepts, \( R \) and \( S \) are roles and \( a \) and \( b \) are individuals:

\[
<\text{axiom}> ::= \text{C} \subseteq \text{D} | \text{C} \equiv \text{D} | \text{R} \subseteq \text{S} | \text{R} \equiv \text{S} | \text{Trans(R)} | \text{C(a)} | \text{R(a,b)} | a = b | a \neq b
\]

An axiom \( \text{C} \subseteq \text{D} \) is called a concept inclusion axiom; it specifies that all instances of the concept \( C \) are also instances of the concept \( D \). The axiom \( \text{C} \equiv \text{D} \) is a concept equivalence axiom, specifying that each instance of \( C \) is an instance of \( D \) and vice versa. Similar definitions hold for the role inclusion and equivalence axioms \( \text{R} \subseteq \text{S} \) and \( \text{R} \equiv \text{S} \), respectively. The axiom \( \text{C(a)} \) is a concept assertion and it specifies that the individual \( a \) is a member in \( C \). The axiom \( \text{R(a,b)} \) is a role assertion and it specifies that individuals \( a \) and \( b \) are connected through role \( R \). Finally, \( a = b \) and \( a \neq b \) specify that individuals \( a \) and \( b \) are equal or different, respectively.

We later give a more detailed example explaining how description logics can be used to model the domain of TUI and TC.

### 3.2.2 Semantics

The semantics of description logic knowledge bases is given by the interpretation \( I \), which is a pair \((\Delta^i, ^1)\) where \( \Delta^i \) is an arbitrary non-empty countable domain set and \( ^1 \) is an interpretation function. Since SHOIQ(D) description logic provides features for modelling concrete predicates, we need an interpretation of the concrete domain as well. It is defined by an arbitrary concrete domain set \( \Delta_D \) and by assigning to each n-ary predicate \( d \) an interpretation \( d^D \subseteq \Delta_D^n \).

The interpretation function \( ^1 \) maps concepts (atomic and defined) to a subsets of \( \Delta^i \), roles to relations over \( \Delta^i \) (i.e. to subsets of \( \Delta^i \times \Delta^i \)), and individuals to members of \( \Delta^i \).

For a fixed interpretation of the concrete domain predicates, the interpretation of the defined concepts must obey the following constraints:

\[
\begin{array}{|c|c|}
\hline
\neg C & \Delta^i \setminus C^i \\
\hline
C \cap D & C^i \cap D^i \\
\hline
C \cup D & C^i \cup D^i \\
\hline
\exists R.C & \{ x \mid (x,y) \in R^i \land y \in C^i \} \\
\hline
\forall R.C & \{ x \mid (x,y) \in R^i \rightarrow y \in C^i \} \\
\hline
\leq n R.C & \{ x \mid \{ y \mid (x,y) \in R^i \land y \in C^i \} \leq n \} \\
\hline
\geq n R.C & \{ x \mid \{ y \mid (x,y) \in R^i \land y \in C^i \} \geq n \} \\
\hline
\end{array}
\]
In an interpretation $I$, we say that axioms are satisfied if the following holds:

<table>
<thead>
<tr>
<th>$C \subseteq D$</th>
<th>$C^I \subseteq D^I$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C \equiv D$</td>
<td>$C^I \equiv D^I$</td>
</tr>
<tr>
<td>$R \equiv S$</td>
<td>$R^I \subseteq S^I$</td>
</tr>
<tr>
<td>$R \subseteq S$</td>
<td>$R^I \subseteq S^I$</td>
</tr>
<tr>
<td>$C(a)$</td>
<td>$a^I \in C^I$</td>
</tr>
<tr>
<td>$R(a,b)$</td>
<td>$(a^I, b^I) \in R^I$</td>
</tr>
<tr>
<td>$a = b$</td>
<td>$a^I = b^I$</td>
</tr>
<tr>
<td>$a \neq b$</td>
<td>$a^I \neq b^I$</td>
</tr>
</tbody>
</table>

If all axioms from a knowledge base $KB$ are satisfied in an interpretation $I$, then $I$ is a model of $KB$. $KB$ is satisfiable if it has a model. Checking satisfiability is the fundamental inference for description logic. Other interesting inferences can be reduced to satisfiability checking, where $\alpha$ is a new individual, previously not occurring in $KB$:

- Concept satisfiability. A concept $C$ is satisfiable if it has a non-empty model. This is the case if and only if $KB \cup \{ C(\alpha) \}$ is satisfiable.
- Concept subsumption. A concept $D$ subsumes a concept $C$, written $KB \models C \subseteq D$, if and only if in each model $I$ of $KB$, $C^I \subseteq D^I$ holds. This is the case if and only if $KB \cup \{ (C \cap \neg D)(\alpha) \}$ is unsatisfiable.
- Concept instantiation. Individual $a$ is an instance of the concept $C$, written $KB \models C(a)$, if and only if in each model $I$ of $KB$, $a^I \in C^I$ holds. This is the case if and only if $KB \cup \{ \neg C(a) \}$ is unsatisfiable.

### 3.2.3 Description Logics and OWL

With the advent of the Semantic Web, it soon became clear that an expressive language for modelling web ontologies is needed. OIL and DAML+OIL [4] are the first languages which tried to use description logics in the field of the Semantic Web. The results from DAML+OIL were incorporated with some minor changes into the design of OWL, the current W3C recommendation for ontology modelling in the Semantic Web [5]. Currently, the following three dialects of OWL exist:

- **OWL-Full** is a logic compatible with RDF(S). Its main feature is that it allows treating the same symbol as a concept and as an instance, by means of which it attempts to mimic second-order logic. Currently it is not known whether OWL-Full is decidable or not.
• OWL-DL is actually a notational variant of the SHOIN(D) description logic and is discussed next.

• OWL-Lite is actually a notational variant of the SHIF(D) description logic and has been designed to allow for easier implementation of reasoning systems.

In the rest of this document, we mainly focus on OWL-DL and OWL-Lite, since they may be summarized as a description logic language with an XML syntax. This syntax is somewhat unwieldy, and does not add much to the understanding. Hence, we refrain from repeating the syntax and direct the reader to [5] for details. In order to enable easier modelling, apart from the traditional DL constructs, OWL-DL and OWL-Lite offer certain shortcuts which are useful for conceptual modelling. These shortcuts are listed in the table below, along with the encoding of the shortcut in description logic.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C ) is the domain concept of role ( R ).</td>
<td>( \exists R.T \subseteq C )</td>
</tr>
<tr>
<td>( C ) is the range concept of role ( R ).</td>
<td>( T \subseteq \forall R.C )</td>
</tr>
<tr>
<td>Role ( R ) is functional.</td>
<td>( T \subseteq \leq 1 R )</td>
</tr>
<tr>
<td>Role ( R ) is inverse functional.</td>
<td>( T \subseteq \leq 1 R^{-} )</td>
</tr>
<tr>
<td>Concepts ( C_1, \ldots, C_n ) are disjoint.</td>
<td>( C_i \subseteq \neg C_j )</td>
</tr>
</tbody>
</table>

3.2.4 Example

The formal syntax and semantics of description logics may be daunting. To aid understanding, in this section we explain how it can be used for solving the problem presented in Section 2.

To begin with, description logics can be very useful for modelling schemata of INT, TUI and TC. For example, in [6] it was demonstrated that entity-relationship schemata can be captured in description logic ontologies. Hence, we first show how to model the conceptualization of TUI, presented in Figure 2, in DL.

We introduce the following atomic concepts, with the specified intuitive meaning. To distinguish names belonging to the description of TUI, we prefix all names with \texttt{tui:}, mimicking the syntax of XML namespaces.

• \texttt{tui:Hotel} – The set of all hotels.
• \texttt{tui:RoomType} – The set of all types of rooms. Individuals in this set are \texttt{tui:single}, \texttt{tui:double} and \texttt{tui:suite}.
• \texttt{tui:Rate} – The set of rates that a hotel offers.
• \texttt{tui:Country} – The set of all countries in the world.
• \texttt{tui:City} – The set of all cities in the world.
• \texttt{tui:Facility} – The set of all facilities. Individuals in this set might be \texttt{tui:parking}, \texttt{tui:swimming-pool} etc.
• \texttt{tui:CreditCard} – The set of all credit card types accepted by TUI. Individuals in this set might be \texttt{tui:MasterCard} or \texttt{tui:Visa}.
Similarly, we use the roles to connect the concepts, as presented in Figure 2. For each role, we include a domain and a range constraint declaration. For example, the role `tui:accepts` will have the concept `tui:Hotel` as the domain and the concept `tui:CreditCard` as the range.

To aid the browsing of the database of all hotels, we introduce some additional definitions which resemble in their functionality to database views. The following definition defines a concept `ExpensiveHotel` as a hotel which offers a rate which has an amount larger than 200 EUR:

\[
\text{tui:ExpensiveHotel} \equiv \text{tui:Hotel} \land \exists \text{tui:offers}.(\exists \text{tui:amount}.>200)
\]

The following definition defines a `tui:EnglishHotel` as a hotel located in a city which is located in England.

\[
\text{tui:EnglishHotel} \equiv \text{tui:Hotel} \land \exists \text{tui:located-in}.(\exists \text{tui:is-in}.(\text{tui:England}))
\]

Information about concrete hotels can now be represented by instantiating the concepts and roles. For example, the following is a snippet of information about such a hotel.

```plaintext
\text{tui:Hotel(h1)}
\text{tui:name(h1,'Central')}
\text{tui:accepts(h1,tui:MasterCard)}
\text{tui:accepts(h1,tui:VISA)}
\text{tui:offers(h1,r1)}
\text{tui:Rate(r1)}
\text{tui:amount(r1,160)}
\text{tui:from-date(r1,'01.02.2004')}\]
\text{tui:to-date(r1,'31.04.2004')}
\text{tui:offers(h1,r2)}
\text{tui:Rate(r2)}
\text{tui:amount(r2,220)}
\text{tui:from-date(r2,'01.02.2004')}\]
\text{tui:to-date(r2,'31.05.2004')}
```

To demonstrate the added expressivity of description logic, consider the case when information about a hotel is incomplete. Hence, it might be that we only know that some hotel `h2` accepts some credit card, but we do not know exactly which one. This might be expressed with the following assertion:

\[
\exists \text{tui:accepts.tui:CreditCard(h2)}
\]

The above statement means that for `h2`, there must be some credit card which is accepted, but we do not know exactly which.

Similarly, a constraint that `h2` does not accept credit cards can be expressed in the following way:

\[
\leq 0 \ \text{tui:accepts(h2)}
\]

The above constraint states that there may be at most zero number of `tui:accepts` links pointing from the object `h2`. If an assertion `tui:accepts(h2, tui:MasterCard)` is accidentally added, the knowledge base would become unsatisfiable.
Let us perform now the same process for the schema of TC from Figure 3. We use the prefix `tc:` to distinguish the elements of the TC’s conceptualization. Similarly to the case for the TUI schema, we introduce an atomic concept for each bubble from Figure 3. We now axiomatize relationships between various concepts.

\[
\begin{align*}
tc:Hotel & \subseteq tc:Accommodation \\
tc:Bungalow & \subseteq tc:Accommodation \\
tc:HotelFacility & \subseteq tc:RoomFacility
\end{align*}
\]

As previously, we may describe a particular hotel using the conceptualization of TC.

In order to integrate data of TC and TUI, INT needs to create an integrated conceptualization. However, there are problems: the granularity of data in both sources is different. Hence, one may choose the ‘intersection’ or ‘union’ approach to integration. In the first case, the integrated conceptualization contains only information found in all the sources. In this way integrated information is always consistent. However, since the sources are heterogeneous, it is realistic to expect that this intersection will be fairly small. Hence, an approach that does not cut off the existing information is the ‘union’ approach, in which the union of information from all sources is preferred.

For example, the TUI conceptualization does not contain the information about the chain to which the hotel belongs. Similarly, it does not contain the information about the standard amount for the rate. Furthermore, the notions of `tui:RoomType` and `tc:RoomType` have the same name, but are semantically different. If we choose to include this information into the integrated conceptualization, we need to determine what data to show for the individuals where this data is now known.

To simplify this example, we assume that INT uses the same conceptualization as TC, with the differences as summarized in Figure 4. In another words, INT takes over the semantics of TUI for `RoomType` and renames TC’s `RoomType` into `RoomClass`.

The next step is to specify the correspondences between the source conceptualizations and the integrated schema. We can do this by using so called inter-schema assertions [7]. These may be viewed as the usual inclusion axioms specifying how the elements of the integrated conceptualization correspond to elements of the source conceptualization.

For example, one may specify that `int:Hotel` includes both `tui:Hotel` and `tc:Hotel` elements in the following way:

\[
\begin{align*}
tui:Hotel & \subseteq int:Hotel \\
tc:Hotel & \subseteq int:Hotel
\end{align*}
\]

Specifying that `int:RoomType` has the semantics of `tui:RoomType`, whereas `int:RoomClass` has the semantics of `tc:roomType`, can be done in this way:

\[
\begin{align*}
tui:RoomType & \equiv int:RoomType \\
tc:RoomType & \equiv int:RoomClass
\end{align*}
\]
We now get to the trickier part of mapping the \texttt{tui:Rate}. Please observe that the conceptualization INT contains more information about a rate. Hence, the assertions mapping TUI’s information space to INT must somehow reflect this missing information.

\[
tui:Rate \subseteq \text{int:Rate} \land \\
\text{\texttt{\leq 0 int:original-amount \land}} \\
\exists \text{int:for-class int:RoomClass} \\
tui:amount \subseteq \text{int:reduced-amount} \\
tui:from-date \subseteq \text{int:from-date} \\
tui:to-date \subseteq \text{int:to-date} \\
tui:for-type \subseteq \text{int:for-type}
\]

The first statement specifies that all rates from TUI are rates in INT. However, all such rates in INT do not have an original amount and are for some room class (but we do not know for which one). In this way, we represent explicitly the fact that some information is missing. Furthermore, we represent explicitly that each TUI rate is for some room class, but we do not know which one. We create similar assertions for the TC source:

\[
tc:Rate \land \exists tc:no-beds.(1) \subseteq \\
\text{int:Rate} \land \exists \text{int:for-type.( int:single )} \\
tc:Rate \land \exists tc:no-beds.(2) \subseteq \\
\text{int:Rate} \land \exists \text{int:for-type.( int:double )} \\
tc:RoomType \subseteq \text{int:RoomClass} \\
\]

In the conceptualization of TC, the number of beds of a room is represented as an attribute of the \texttt{tc:Rate} concept. However, in the conceptualization of INT, the number of beds is represented as an instance of the concept \texttt{int:RoomType}. Hence, the first two assertions separate the different values of the \texttt{tc:no-beds} attribute and replace them with an appropriate individual.

Inferences can now be used to answer queries over INT. For example, one may ask the following question to retrieve all hotels offering a rate for a double room with the price

\begin{figure}
\begin{center}
\includegraphics[width=\textwidth]{int_conceptualization.png}
\end{center}
\caption{Differences in the Conceptualization of INT}
\end{figure}
of less than 100 EUR”

\[ \text{int:Hotel} \land \\
\exists \text{int:offers.}( \\
\exists \text{int:for-type.} \{ \text{int:double} \} \land \\
\exists \text{int:reduced-amount.}<100\})(X) \]

3.3 F-Logic

F-Logic [8] is a logical formalism targeted mainly to specification and manipulation of object-oriented logical databases. It is important to understand that F-Logic, as presented in [8], is not just one language, but a family of languages. For the lack of a better name, we call the fundamental language F-Logic/logic. This language is a proper extension of first-order logic, and it does not provide non-monotonic features, such as default inheritance, type checking of well-founded negation. Instead, F-Logic/logic provides a sound and complete refutation procedure for checking consistency of F-Logic knowledge bases.

F-Logic/logic is not itself meant for practical usage. Rather, its main role is to serve as a basis for F-Logic/LP – a logic programming variant of F-Logic. This variant of the language is implemented in current tools, such as Flora [10], Ontobroker [9] or Triple [11]. The relationship between F-Logic/logic and F-Logic/LP is roughly the same as the relationship between first-order logic and Prolog: the first language sets the stage, whereas the second language implements a subset of the first language and adds numerous practical features. Since all existing tools we know of implement the F-Logic/LP variant, we shall focus in this document to present only this practical variant of the language. Furthermore, the formal syntax and semantics of F-Logic as presented in [8] is quite complicated. Hence, we refrain from a formal presentation, but give a more practical overview based on numerous examples.

3.3.1 Objects and Classes in F-Logic

All entities in F-Logic are identified by means of an object identifier (OID). An OID corresponds to a term in first-order logic. It can either be a symbol, a variable or a functional term. An OID with variables is meant to represent all identifiers obtained by replacing variables with some values. We employ the usual syntactic convention of Prolog where variables are written with their first letter in uppercase. Examples of OIDs are:

- person, X, f(X)

F-Logic was inspired by the object-oriented modelling style. Hence, F-Logic knowledge bases consist primarily of objects and classes, each of them identified by OIDs. As usual in object oriented systems, in F-Logic classes can be arranged in a class hierarchy. Syntactically, the fact that the class \( sc \) is a subclass of the class \( c \) is specified in the following way:

\[ sc::c. \]

Furthermore, specifying that the object \( o \) is an instance of the class \( c \) can be done in the following way:

\[ o::c. \]
However, the distinction between objects and classes is not strict in F-Logic. The same symbol can be used to denote an object and a class at the same time. A classical example demonstrating the need for such modelling is given in [12]. Consider the relationship between the notions of species, ape and a concrete ape ape1. A possible view of the world is that ape is an instance of species, and that ape1 is an instance of ape. Hence, ape is a class and an instance at the same time. This can be modelled in F-Logic in the following way:

```
ape: species.
ape1: ape.
```

As usual in object-oriented systems, objects in F-Logic can have methods. Invocation of a method is denoted by the specifying the method’s name in square brackets after the name of the object. For example, specifying that the person peter is 30 years old can be done in the following way:

```
peter[age->30].
```

The name and the value of a method are OIDs, which can contain functional terms. Hence, a method with parameters can be represented in the following way (in some syntactical variants of F-Logic, the method name must be delimited from the method parameters using the sign @):

```
peter[ih_child(1)->paul].
peter[ih_child(2)->mary].
```

There are essentially two types of methods: single-valued methods and set-valued methods. Invocation of a single-valued method is denoted using the connective ->, whereas invocation of a set-valued is denoted using the connective ->>. There is a significant different between the usage of -> and ->>: namely, from a[m->v1] and a[m->v2] one can derive v1=v2, which does not hold if methods are invoked using the connective ->>. In this respect, single-valued methods are similar to functional properties of description logics.

The syntactical unit o[m->v, ..., ] is called an F-molecule and it plays the role of atoms in first-order logic. F-molecules can be used to form F-Logic formulae using the usual and (∧), or (∨) and not (¬) connectives and existential (∃) and universal (∀) quantifiers. In F-Logic/logic, one can build arbitrary formulae, just like in first-order logic. In F-Logic/LP, similarly to how it is done in Prolog, only Horn formulae, possibly with disjunction in the rule body, and with all variables universally quantified, can be used. For example, the following rule specifies that a child has the same last name as his father:

```
X[lastName->LN] :- X:child[father->F], F[lastName->LN].
```

In F-Logic/LP, such formulae are usually called rules. Most tools allow for using non-monotonic negation in rule bodies, which is in the Flora version denoted with tnot (the actual syntax for non-monotonic negation varies from tool to tool). For example, saying that a person not married to someone is single can be done in the following way:

```
X[marrital_status->single] :- X:person,
    tnot X[married_to->M].
```
It is well-known that for non-monotonic negation there are several different semantics. In F-Logic, well-founded semantics has been taken as standard [13], since it allows a high level of expressivity, while retaining tractability.

3.3.2 Type System

Apart from merely specifying objects and classes, F-Logic provides a type system for checking arguments of methods and their return values. Specifying that the value of the method married_to for the object peter must belong to class person can be done in the following way:

\[ \text{peter\{married\_to } \Rightarrow \text{ person\}.} \]

It is important to understand that correctness of type declarations in an F-Logic knowledge base is not built into the F-Logic language. Rather, correctness can be checked by specifying the following pair or rules:

\[
\text{scalar\_type\_error(O,M,R,D) :-}
\begin{align*}
& \quad O[M->R], \ O[M=>D], \ \text{tnot R:D;}
& \quad O[M->R], \ \text{tnot O[M=>_D].}
\end{align*}
\]

\[
\text{set\_type\_error(O,M,R,D) :-}
\begin{align*}
& \quad O[M->R], \ O[M=>>D], \ \text{tnot R:D;}
& \quad O[M->R], \ \text{tnot O[M=>>_D].}
\end{align*}
\]

Checking a knowledge base for type errors can be done by a query to both \text{scalar\_type\_error(O,M,R,D)} and \text{set\_type\_error(O,M,R,D)} rules. In both rules, the first body clause checks for values of incorrect type, whereas the second clause checks for values whose type has not been declared. Please note that \text{_D} is a variable distinct from \text{D}.

3.3.3 Inheritance

Since it is inspired by object-oriented systems, F-Logic places a great deal of attention to method inheritance. These features are specific to F-Logic/LP and are not available in F-Logic/logic. This is so because F-Logic/logic is an extension of first-order logic and as such does not offer non-monotonic features. Inheritance is, however, a clearly non-monotonic feature.

Until now, methods have been declared for one object only, i.e. at an instance, and not at the class level. However, in F-Logic it is possible to specify a default value of a method for a class of objects. The following example shows how the typical default inheritance problem can be modelled in F-Logic: all birds fly so ‘normal’ birds, such as sparrows do fly, but penguins do not fly.

\[
\begin{align*}
\text{bird}\{\text{flies } & \Rightarrow \text{ yes}\}. \\
\text{sparrow:bird.} \\
\text{penguin:bird}\{\text{flies } & \Rightarrow \text{ no}\}.
\end{align*}
\]

The question “what birds are known, and do they fly?” gives the following answers:

\[
?\quad \text{B:bird}\{\text{flies}\Rightarrow\text{A}\}. \\
\text{B } = \text{ sparrow} \\
\text{A } = \text{ yes}
\]
B = penguin
A = no

The semantics of default inheritance operator *-> can best be understood in the following way: C[method*->value] and O:C |= O[method->value] if O does not define an override value for method. The specification how conflicts are resolved in case when there are multiple values is quite complex, so we do not elaborate on it further in this document.

Values declared with *-> are propagated to subclasses. For example, if a prey_bird is declared as a subclass of the class bird by the axiom prey_bird:bird, then one can derive prey_bird[flies *-> yes]. Also, one can declare the whole class of flightless birds and declare an ostrich to be a flightless bird in the following way:

flightless_bird::bird[flies *-> no].
ostrich:flightless_bird.

From these declarations one can now derive ostrich[flies -> no].

Inheritance rules for *=> are similar to the ones for *->. Specifying that for instances of a class person the method married_to can have a value of class person can be done in the following way:

person[married_to *=> person].

Now from peter:person one can derive peter[married_to=>person], which means that peter can be married only to objects who are instances of the person class.

3.3.4 Example

We now show how the travelling example from Section 2 can be modelled in F-Logic. The overall modelling principles are similar to those from Section 3.2.4 with a straightforward adaptation of the syntax. Since the symbol : is reserved in F-Logic for specifying class membership, we separate the namespace prefix by #. An excerpt of declarations for the schema is given next:

tui#hotel{
    tui#name *=> string,
    tui#no_stars *=> integer,
    tui#accepts *=>> tui_credit_card,
    ...
}.

The schema of TC can be specified in a similar way:

tc#hotel:tc#accomodation{
    tc#name *=> string,
    tc#is_part_of *=> integer].
    tc#located_in *=> tc#city_area,
    ...
}.

Finally, the integrated schema of INT is specified in a similar way:

int#hotel:int#accomodation{
    int#name *=> string,
Concrete instances of hotels at various providers can now be specified in the following way:

\[
\begin{align*}
\text{h1:tui#Hotel} & : \{ \\ tui\#name & -> 'Central', \\
& tui\#accepts -> \{ tui\#master_card, tui\#visa \}, \\
& tui\#offers -> \{ r1,r2 \}, \\
& \ldots \\
\text{r1:tui#rate} & : \\ tui\#amount & -> 160, \\
& tui\#from_date -> '01.02.2004', \\
& tui\#to_date -> '31.04.2004' \\
\}
\end{align*}
\]

Correspondences between schemata can be now expressed using rules. Specifying that all hotels from TUI and TC are hotels in INT can be done in the following way:

\[
\begin{align*}
\text{X:int#hotel} :& : \text{X:tui#hotel}; \text{X:tc#hotel}.
\end{align*}
\]

As before, the trickier aspect of specifying correspondences is related to specifying correspondences between rates. We may observe several technical problems. Namely, the rate at TUI does not contain information about the room class. Hence, one needs to ‘invent’ a virtual room class when mapping the information. Furthermore, in F-Logic it is not possible to specify that, for such a virtual room, the original_amount the rate must be smaller than 0; rather, comparison is allowed only for concrete data.

\[
\begin{align*}
\text{X:int#room_type} :& : \text{X:tui#room_type}.
\end{align*}
\]

Similarly, the types of rates of TC can be mapped using the following rules:

\[
\begin{align*}
\text{X:int#rate} :& : \text{X:tui#rate}[ \\ int\#reduced_amount & -> A, \\
& int\#from_date -> FD, \\
& int\#to_date -> TD, \\
& int\#for_type -> FT, \\
& int\#for_class -> f(X) \\
] : \text{X:tui#rate}[ \\ tui\#amount & -> A, \\
& tui\#from_date -> FD, \\
& tui\#to_date -> TD, \\
& tui\#for_type -> FT \\
] \\
\text{X:int#rate} :& : \text{X:tc#rate}[:tc:no_beds & -> 1].
\end{align*}
\]
```
X: int#rate[int#for_type -> int#double] :-
  X: tc#rate[tc:no_beds -> 2].
```

Now usual inferences of F-Logic can be used to answer queries over INT. For example, all hotels offering a rate for a double room with the price of less than 100 EUR can be found in the following way:

```
?- X: int#hotel[int#offers->R],
   R: rate[int#for_type->int:double, int#reduced_amount->A],
   A<100.
```

### 3.4 Conceptual Modelling

As ontologies grow in size and in practical relevance, it is legitimate to question whether database techniques could provide interesting support for ontology management. On the one hand, database systems are know to offer scalable and efficient management of huge amounts of structured data, which is what ontologies may become in the near future. On the other hand, conceptual modelling approaches (that have specifically been designed to support a semantically rich description of structured data sets) could at least to some extent handle the description of the conceptualization that is the subject of an ontology. Exploring this idea is definitely worth a discussion in this report.

Arguments in favour of ‘highly intuitive’ ontology models, with a ‘frame-like look and feel’ or ‘database schema’ alike, have already been developed in e.g. [36], [33] and [35]. Specific proposals include [34] and [32]. Mappings from conceptual models to description logics have been proposed in e.g. [30] and [31].

#### 3.4.1 Overview

The following brief description of conceptual modelling expressiveness is based on work on extended entity-relationship (EER) models, which are most frequently seen as offering the richest semantic expressiveness. In EER models and alike, data structures are basically graphs of object types interconnected by relationship types. Both object and relationship types may be characterized by associated properties (attributes and methods). Attributes may be atomic (as in relational tables) or composed of other attributes, thus allowing the definition of multilevel property trees for object and relationship types. It is then possible to represent a real world entity as a single object in the database. Attribute cardinalities state whether an attribute is optional or mandatory, and if it is mono- or multi-valued (list, set, or bag). Relationship types connect object types via roles. A relationship type may be defined with 2 (for binary relationships) to n roles. When two or more roles connect to the same object type, the relationship type is said to be cyclic. Relationship types may be adorned with specific semantics, of which the most well known is aggregation semantics (expressing that an object is a component of another object).

Objects and relationships bear a system-defined, unique identity. Object types and relationship types may be organized into generalization/specialization lattices using is-a links. Inheritance, refinement, redefinition and overloading mechanisms apply as proposed in traditional object-oriented data models. Some advanced conceptual models, however, depart from object-oriented rules by adopting a multi-instantiation paradigm, i.e. allowing the same real world entity to be simultaneously represented by several instances in different classes that are not in a sub-type/super-type relationship. Allowing multi-instantiation is necessary from the modelling point of view, in particular to be able to properly describe situations such as, for example, a real world object being at the
same time a hotel and a restaurant (assuming Hotel and Restaurant are two object types) without requiring a so-called intersection class Hotel&Restaurant, which is a subclass of both Hotel and Restaurant. Another facility from some conceptual model is classification dynamicity, i.e. the possibility for an instance to move to another class (e.g. a guesthouse can become a hotel; a student can become a faculty).

An interesting feature of conceptual models is that they can be extended beyond the description of data structure aspects to cover additional modelling dimensions, such as space (for geographic data representation), time (to support past and future information), combination of space and time (to represent spatio-temporal phenomena such as e.g. mobile objects and trajectories [37][39] and multi-representation (to support content-dependent information) [40]. These extensions typically include the definition of specific data types (such as point, line, and area to describe space extents, instant and interval to describe time extents) with their associated methods, and rules to associate these data types to objects, relationships and attributes.

Conceptual models come with formal definitions, rules to translate conceptual specifications into logical level specifications, and implementations in several marketed CASE tools and in research prototypes.

3.4.2 Example

The TUI conceptualization from Figure 2 can be easily reformulated in an EER formalism by using the following very simple (but not very intelligent) rules:

- ignore links labelled instanceOf and their source ellipsis (EER schemas do not describe instances),
- each ellipsis translates into an object type,
- each link between ellipses translates into a binary relationship named after the label associated to the link,
- each label not in an ellipsis translates into a property (an attribute) of the object type it is linked to.

![Figure 5. Straightforward Reformulation of Figure 2 using EER Formalism](image-url)
Figure 5 shows the diagram for this EER schema.

However, such a schema definition, although syntactically correct, is not really what best defines a conceptual schema for the TUI conceptualization. Modelling RoomType as an object type is questionable, as it would only hold three instances (single, double, and suite). More appropriately, what type of rooms a hotel offers is modelled by attaching to the Hotel object type a multi-valued attribute roomType, with an enumerated value domain equal to \{single, double, suite\}.

Let us now consider rates. Clearly, rates are attached to hotels or hotel rooms. They do not have any independent existence; so modelling rates as object types is also not appropriate. Depending on the perception one wants to represent, rates may be modelled, for example, as a multi-valued attribute of hotel. This would be a composite attribute, with component attributes amount, fromDate, toDate, and roomType. Alternatively, one could model rates as a multi-valued attribute of the roomType attribute. In this case, roomType is turned into a multi-valued composite attribute roomTypes, with two component attributes: a mono-valued attribute type (holding one value from the set \{single, double, suite\}), and a multi-valued composite attribute rates, with component attributes amount, fromDate, and toDate.

Similarly, modelling Facility as an object type is only useful if one wants to independently keep, query, and maintain a catalogue of facilities. If not, facilities can more simply be modelled as a multi-valued attribute of Hotel, with an enumerated domain containing all possible values. The same holds for CreditCard.

Assuming TUI wants to keep a catalogue of cities and countries, the conceptual schema for the TUI conceptualization reduces to the one illustrated in Figure 6, with Figure 7 showing the two options for the attributes of Hotel (please note that in the notation we use here indenting is used to visualize attribute composition). The figures show cardinality constraints (on roles of relationship types and on attributes), as these are traditionally included in an EER schema definition. In Figure 6 we have assumed that a country includes many cities and a city has many hotels, while a hotel may only be located in one city (but some hotels are in no city) and a city is located in only one country. Figure 7 assumes that a hotel may have rooms of different types, many facilities, accept several credit cards, offer many rates (a rate possibly holding for different types of rooms, e.g. a hotel having the same rate for single and double rooms, and possibly many rates holding for the same type of room, e.g. a week rate and a week-end rate).

Notice that in the Hotel description in Figure 7(a), roomTypes appears twice. The first roomTypes attribute provides knowledge on which types of rooms the hotel offers, while the second roomType attribute specifies for which room types a given rate is valid (some hotels may offer the same rate for single and double rooms). Redundancy in the stored data may be avoided by defining the first roomType as a derived attribute, whose value is computed from the values of the roomTypes components of rates. Derived attributes (as well as derived relationships and derived objects) may be supported.

![Figure 6](image-url)
The same reasoning scheme may be applied to produce an EER schema for the TC conceptualization shown in Figure 3. The structural difference is that the TC diagram shows some is-a links that will be translated as is-a links (rather than as relationship types) in the EER design. This refers to the links between Hotel and Accommodation, Bungalow and Accommodation. It may also refer to the links between HotelFacility and Facility, and RoomFacility and Facility, in case Facility holds a catalogue and is therefore modelled as an object type (rather than an attribute). If this is not the case, i.e. Facility becomes an attribute of the Hotel object type. Keeping the difference between hotel facilities and room facilities requires the ability to define the domain of the facilities attribute as a hierarchical value domain. A hierarchical value domain holds several levels of values, organized into a hierarchy of values. In this example, the domain for facilities would hold at the first level the two values, hotel-facility and room-facility, and would at the second level hold two sets of value, one corresponding to the hotel-facility value (e.g. bar, fitness room, dancing, swimming pool etc.) and the other one corresponding to the room-facility value (e.g. minibar, TV, heating, air-con, in-room coffee). Hierarchical value domains are a common feature in geographical data handling.

A possible schema diagram for the TC conceptualization is shown in Figure 8. We assume that, as for TUI, the TC items in Figure 3 that are not shown in Figure 8 have been turned into attributes of the Hotel object type.

Bridging the semantic differences between the TUI and TC descriptions requires precise identification of how things modelled in one description correspond to things modelled in the other description. Such mapping knowledge is expressed as inter-schema corresponding assertions, for which we give below the format suggested in [38].

For example, assuming the Hotel concept is the same in TUI and TC, and assuming TUI and TC providers may have offers for the same hotel, an assertion may be:

<table>
<thead>
<tr>
<th>Hotel</th>
<th>Hotel</th>
</tr>
</thead>
<tbody>
<tr>
<td>name (1:1)</td>
<td>name (1:1)</td>
</tr>
<tr>
<td>no.stars (1:1)</td>
<td>no.stars (1:1)</td>
</tr>
<tr>
<td>no.rooms (1:1)</td>
<td>no.rooms (1:1)</td>
</tr>
<tr>
<td>roomTypes (1:n)</td>
<td>facilities (0:n)</td>
</tr>
<tr>
<td>facilities (0:n)</td>
<td>creditCards (0:n)</td>
</tr>
<tr>
<td>creditCards (0:n)</td>
<td>roomTypes (1:n)</td>
</tr>
<tr>
<td>rates (1:n)</td>
<td>type (1:1)</td>
</tr>
<tr>
<td>amount (1:1)</td>
<td>rates (1:n)</td>
</tr>
<tr>
<td>fromDate (1:1)</td>
<td>amount (1:1)</td>
</tr>
<tr>
<td>toDate (1:1)</td>
<td>fromDate (1:1)</td>
</tr>
<tr>
<td>roomTypes (1:n)</td>
<td>toDate (1:1)</td>
</tr>
</tbody>
</table>

Figure 7. Two Possible Conceptualizations of Hotel Attributes of TUI
The first line asserts that Hotel object types in TUI and TC describe overlapping sets of real world entities, i.e. the same hotel may be instantiated in both TUI and TC. The second line (where the acronym WCI stands for With Corresponding Identifiers) states that two instances in TUI.Hotel and TC.Hotel represent the same real world hotel if the value for the name attribute is the same in the two instances. This provides knowledge about how to identify in one data set the instance that corresponds to a given instance in the other data set. Thanks to this knowledge, it becomes possible to gather all information about a real world thing that is available in the various interrelated sources. The following lines, introduced by the WCA (With Corresponding Attributes) acronym, allow defining a corresponding attributes for the related types, i.e. attributes that at least partially represent the same real world property, irrespectively of how it is coded in the representation. The first WCA line, for example, states that for corresponding instances the creditCard attributes hold identical information in both TUI and TC. The second WCA line states that facilities in TUI and TC denote the same type of information (same concept) but may show different values even for corresponding instances (for the same hotel, facilities recorded in TUI may not be exactly the same as facilities recorded in TC). Hence, to retrieve the facilities offered by a given hotel, both TUI and TC have to be searched. Finally, the third WCA line states that although the two rates attributes hold the same type of information, the values in TUI and TC are different (i.e. a room rate is always a room rate, but TUI and TC do not talk about the same rates even for the same hotel).

Generically speaking, assuming two related attributes A and B, attribute correspondence assertions (WCA) may be any one of A=B (same value), A=f(B) (the value of A may be computed from the value of B), A≠B (different values). If A and B are multi-valued attributes, additional possibilities are A ⊆ B (values in B are a subset of values in A), and A ∩ B (A and B have overlapping collections of values). The same extent of possibilities holds for correspondences between elements (e.g. TUI.Hotel and TC.Hotel).

![EER Design for the TC Conceptualization](image)
Many of these assertions would be necessary to describe all correspondences between the TUI and the TC sources. From these assertions, the INT view of the available data may be built almost automatically. ‘Almost’ refers here to the fact that many alternative INT views may be elaborated from the same set of assertions, depending on designer’s preferences (e.g. one designer may prefer a more concise schema while another designer may prefer a more exhaustive schema). This is possible thanks to the semantic relativism that characterizes the modelling activity.
4 Feature Comparison

As can be observed from Section 3, current approaches to ontology modelling are quite heterogeneous. Each approach offers features guided by different requirements. Even to an experienced user, it is often unclear how these features interrelate. The nuances in the formal semantics are often quite intricate and far from obvious. Therefore, in this section we contrast crucial features explicitly and explain their similarities and differences.

4.1 Domains and Ranges

Probably the most common misconception of users when starting to work with OWL-DL is the meaning of domain and range declarations in OWL-DL ontologies. This is caused mainly due to the comparison with object-oriented data models, which are far more widely spread than description logics.

Consider the snapshot from an ontology describing the conceptualization of TUI presented in Figure 9. The meaning of the conceptualization is that there are hotels, there are credit cards and that certain hotels accept zero or more credit cards. Furthermore, h1 is a hotel and it accepts Visa. This conceptualization can be interpreted as the following informal ontology:

- Hotel is a concept,
- CreditCard is a concept,
- accepts is a property,
- accepts has Hotel as domain,
- accepts has CreditCard as range

Such a representation has an intuitive appeal, but is quite ambiguous, as it does not precisely state what is the meaning of the ‘domain’ and ‘range’ modelling constructs. As we shall demonstrate, this semantics differs from between formalisms.

In classical object-oriented data models, the above statements would be interpreted like this: the property accepts can be established only between objects which are instances of the Hotel concept and the CreditCard concept. This means that a property accepts could not have been established between h1 and Visa, unless h1 were known to be a Hotel and Visa to be a CreditCard. Establishing the property between objects which do not satisfy the domain and range constraints would cause a type error. Systems such as object-oriented databases would not allow such an error to occur in the first place – the request to create the accepts property between instances that do not

![Figure 9. Semantics of Domains and Ranges](image)
match the constraints would be refused by the database.

In description logics the situation is quite different. The statements that the `accepts` property has the concept `Hotel` as domain, and the concept `CreditCard` as range are written in description logic syntax like this:

\[
\exists \text{accepts.} T \subseteq \text{Hotel} \\
T \subseteq \forall \text{accepts.} \text{CreditCard}
\]

In particular, the first statement says that anything which has an instance of the `accepts` property pointing to any other object must be an instance of a `Hotel`. The second statement says that for all instances, the `accepts` property may point only to instances of `CreditCard`.

At the first glance, there seems to be no difference in semantics. These statements also state that the `accepts` property can be established only between instances of `Hotel` and `CreditCard`. However, a closer look reveals that this is not at all so. Namely, the usual semantics of object-oriented systems should be more precisely expressed like this: the `accepts` property can be instantiated only between objects which are `known` to be `Hotel` and `CreditCard`. In the description logic interpretation the word ‘known’ is missing. Hence, from an ABox assertion

\[
\text{accepts}(h1, \text{Visa})
\]

one can derive that `h1` must be a `Hotel`, and that `Visa` must be a `CreditCard`. Hence, an ABox specifying `accepts(h1, Visa)` without specifying `Hotel(h1)` and `CreditCard(Visa)` is not incorrect, as in the case of object-oriented systems. Rather, such an ABox entails the missing assertions.

In F-Logic the situation is closer to the semantics of classical object-oriented languages. The statement that the `accepts` property can be established between instances of a `hotel` and `credit_card` can be written as follows:

\[
\text{hotel[accepts} \ast \Rightarrow \text{credit_card]}
\]

It is worth noting that even in F-Logic it is possible to assert `h1[accepts \Rightarrow visa]` without explicitly stating `h1:hotel` and `visa:credit_card`. Current F-Logic tools will successfully read such a knowledge base and perform inferences over it. In order to detect a type error, one should invoke the rules from Subsection 3.3.2. As noted in Subsection 3.3.2, the type system of F-Logic is separated from the core language, and it is up to the language users to worry about type enforcement. In this sense, F-Logic is somewhere in between object-oriented and description logic systems.

One may validly ask the question why description logic semantics departs from the usual semantics of domains and ranges. The answer is that description logics are not so much focused on managing individuals. Rather, they have been optimized for reasoning about concept descriptions and determining the subsumption relationships between them. The semantics of domains and ranges makes perfect sense in this case. For example, let us assume that INT wants to locate new providers of information about hotels. To facilitate the discovery, each provider might create an advertisement specifying which types of hotels he covers. It is natural to model this advertisement as a concept, specifying the set of all potential instances that the provider can provide information about. For example, a provider about hotels which belong only to European chains might create the following advertisement:
The broker which is interested in hotels belonging to European or US chains might create the following query:

\[
\text{Hotels\_QRY} = \text{Hotel} \cap \\
\exists \text{is-part-of.}(\text{EUChain} \cup \text{USChain})
\]

Now searching for potential providers of hotel information can be done by testing whether the intersection of concepts \text{Hotels\_ADV} and \text{Hotels\_QRY} is not empty (which is the case if and only if \(\text{KB} \cup \{ \text{Hotels\_ADV} \land \text{Hotels\_QRY}(a) \} \) is satisfiable, an inference which can be performed efficiently by description logic systems).

For such matching to take place, the semantics of quantifiers in the definitions is clear: for each instance of \text{Hotels\_ADV}, the \text{is-part-of} property must point to an EUChain. If one would use only declarations specifying the allowed database states, such inferences at the intentional level would not be possible.

Let us now assume that the range of the \text{is-part-of} property is declared to be a Chain by means of an axiom \(\exists \text{is-part-of.}\)EUChain. Such a statement can be used for preventing potential errors. For example, the broker might misinterpret the semantics of the \text{is-part-of} property and create a query \text{Hotel} \cap \exists \text{is-part-of.}\)EUCity. However, since the range of \text{is-part-of} is declared to be an EUChain, if EUCity is declared to be disjoint with EUChain, it is possible to detect that the query is inconsistent with the ontological specifications.

There is another explanation why description logic provides exactly this type of semantics. Namely, referring to what the knowledge base knows or not is not possible in first-order logic, as such features are non-monotonic. The creators of OWL-DL deemed it important to follow the semantics of first-order logic, so they have not included any non-monotonic features into the language.

However, description logic has been extended with non-monotonic features. Here we briefly overview the extension of description logic by the autoepistemic operator presented in [14]. This extension can be used to formalize the usual semantics of domain and ranges.

The extension involves defining a new operator, which can be used in building concept and role descriptions. This operator is denoted with \(K\), and its intuitive meaning is to represent the information about concepts and roles which is known to the knowledge base. For a concept \(C\), the interpretation \(KC\) is defined as the set of individuals which are members of \(C\) in all possible worlds. With such a definition, the usual domain and range constraints can be formulated as concepts which must hold for every individual:

\[
C_1 \equiv \neg \exists K\text{accepts.}T \cup K\text{Hotel} \\
C_2 \equiv \forall K\text{accepts.}K\text{CreditCard}
\]

A knowledge base satisfies all constraints if \(C_1\) and \(C_2\) hold for each individual in the knowledge base. Intuitively, these declarations should be interpreted like this: for each individual, either it must be known that it does not have an \text{accepts} property, or it must be known for the individual that it is a \text{Hotel}. Similarly, for each individual, the
The `accepts` property may point only to individuals which are known to be an instance of `CreditCard`.

### 4.2 Closed vs. Open Worlds

There is an important difference in the assumption on the completeness of knowledge of description logics and common logic programming systems. Namely, description logics employ the so-called open-world assumption. This means that in description logic, if something is not explicitly said to be false, it is not assumed to be false. In another words, description logic assumes that the knowledge about the world is incomplete.

On the other hand, the logic programming systems typically assume a closed-world assumption. Under this assumption, if nothing is not explicitly said to be true, it is assumed to be false. In another words, logic programming systems assume that the knowledge about the world is complete, which reflects itself on the semantics of negation.

The open-world assumption of description logic is the consequence of the fact that DL is firmly rooted in first-order logic, which itself employs open-world assumption. The closed-world assumption cannot be axiomatized in first-order logic. Furthermore, whereas the semantics of the open-world assumption is widely accepted and agreed upon, numerous attempts to formalize the closed-world assumption exist, and there is no general consensus.

Finally, it is important to understand that F-Logic/logic, being the extension of first-order logic, also employs the open-world assumption. On the other hand, the F-Logic/LP variant which has been implemented in practical tools follows the path taken by logic programming systems such as Prolog and employs the closed-world assumption.

Even to experienced experts the exact difference between these two views of the world is sometimes obscure. Therefore, we illustrate these differences by means of several examples.

Consider the description logics knowledge base KB based on the conceptualization of TC from Figure 3. In this conceptualization, the property `accepts` links instances of the concept `Hotel` with instances of the concept `CreditCard`. Let us assume that `MajorCreditCard` is a subclass of `CreditCard` with instances `Visa` and `MC` (the latter stands for MasterCard). The knowledge base axioms are as follows:

\[
\exists \text{accepts}. T \subseteq \text{Hotel} \\
T \subseteq \neg \text{accepts}. \text{CreditCard} \\
\text{MajorCreditCard} \subseteq \text{CreditCard} \\
\text{MajorCreditCard}(\text{Visa}) \\
\text{MajorCreditCard}(\text{MC}) \\
\text{Hotel}(h1) \\
\text{accepts}(h1, \text{Visa})
\]

With such a knowledge base, one might want to check whether `h1` accepts `MC`. However, the knowledge base contains neither `accepts(h1, MC)` nor `\neg \text{accepts}(h1, MC)`. Hence, the answer to question $KB \models \text{accepts}(h1, MC)$ is ‘no’. It is now very important to precisely understand the implications of such an answer. Namely, the answer ‘no’ simply means that KB does not entail `accepts(h1,MC)`. This answer does not say
anything about the converse question: this ‘no’ does not mean that $KB \models \neg \text{accepts}(h1,MC)$ holds. In fact, the answer to the latter question is also ‘no’. These two negative answers should be interpreted as ‘KB does not know whether $\text{accepts}(h1,MC)$ or $\neg \text{accepts}(h1,MC)$ holds; there is simply not enough information’.

Consider now the following F-Logic/LP program P, which basically represents the same information as KB:

```
hotel[accepts *=>> credit_card].
major_credit_card::credit_card.
visa:major_credit_card.
mc:major_credit_card.
h1:hotel[accepts -=> visa].
```

Now the answer to the question $? - h1[accepts -=> mc].$ is again ‘no’. However, this ‘no’ has a different meaning: it means that from the information in P, one cannot prove that h1 accepts mc, so we assume that opposite. Hence, the answer to the converse query $? - \neg \text{not h1}[accepts -=> mc].$ is ‘yes’.

This example shows that description logic systems assume that they do not know everything, whereas logic programming systems assume that they know everything, and everything not contained in the program is false. Closely related to these views of the world are the properties of (non-)monotonicity. Namely, description logic is monotonic, which means that by adding new information, one cannot invalidate any positive answer. For example, by adding the fact \text{accepts}(h1,mc)$, some answers may change from ‘no’ to ‘yes’, but no answer can change from ‘yes’ to ‘no’. The latter is the case even if one adds the fact $\neg \text{Hotel}(h1)$. In this case, the resulting knowledge base does not have a model, i.e. it is unsatisfiable, and by definition, everything follows from an unsatisfiable knowledge base.

F-Logic/LP is non-monotonic, since adding new information can change a ‘yes’ answer into a ‘no’ answer. For example, by adding the fact $h1[accepts -=> mc]$, the answer to query $? - \neg \text{not h1}[accepts -=> mc].$ changes from ‘yes’ to ‘no’ (since now we know that h1 accepts MasterCard, so it is not the case that h1 does not accept MasterCard).

There are other important aspects of the open- and closed-world assumption. Imagine someone wanted to know all hotels which accept only major credit cards. This query might be formulated by adding the following axiom to KB, thus resulting in a knowledge base $KB_Q$:

```
\forall \text{accepts}.\text{MajorCreditCard} \subseteq Q
```

Now an important question is whether $KB_Q \models Q(h1)$, i.e. whether h1 accepts only major credit cards. At the first glance, one might assume that the answer is ‘yes’: after all, the knowledge base contains only information that h1 accepts Visa, and Visa is an instance of \text{MajorCreditCard}. However, this is wrong: the knowledge base does not contain information that h1 accepts only Visa. For all the knowledge base knows, h1 may be accepting some non-major credit card, such as Diners.

A similar question may be posed to the F-Logic/LP program P. However, the Flora implementation of F-Logic does not allow specifying such a query directly, since it con-
tains the universal quantifier in the body of the rule. Luckily, the query can be rewritten into a similar one by using the identity $\forall R.C \equiv \neg(\exists R.\neg C)$. Intuitively, this identity states that objects which are linked only to $C$ are those object which are not linked to one element which is not $C$. Hence, the query can be represented using double negation:

$$q(X) \quad :- \quad X:hotel, \ tnot \ X:accepts\_any\_cc. $$
$$X:accepts\_any\_cc \quad :- \quad X:hotel[accepts \rightarrow C], \ tnot \ C:major\_credit\_card.$$

The query $q$ selects all hotels which do not accept any credit card, whereas the latter task is performed by the query $accepts\_any\_cc$. Note that $X:hotel$ in the definition of the rule for $q$ is necessary to bind a variable $X$ to instances of the hotel. Otherwise, the variable could be bound to anything (it flounders). Now the question $?- q(h1)$. returns ‘yes’: $h1:accepts\_any\_cc$ does not hold, so $q(h1)$ does hold.

From this example one may see that open-world and closed-world assumption are closely related to the semantics of negation. Namely, in a language without negation (e.g. in the language of Horn clauses), it is not possible to ask questions which might lead to different interpretations. The notion of a ‘language without negation’ should be taken loosely: the above question does not contain negation explicitly, but the negation is hidden in the actual meaning of the universal quantifier.

Given these considerations, a natural question arises how to bring these two semantics together. In the case of description logics, there is a simple solution to the problem. For $h1$ to be a part of the answer to our query, one should explicitly specify that the information about the knowledge base is complete, which can be done by saying that $h1$ accepts Visa and it accepts at most one credit card, by including the following axiom in the knowledge base:

$$\leq 1 \ accepts\_Credit\_Card(h1)$$

Now the information in KB is complete: in any possible world, $h1$ accepts Visa, $h1$ cannot accept any other CreditCard and Visa is a MajorCreditCard. Hence, in any possible world $q(h1)$ holds. Hence, in description logics it is possible to explicitly close-off a world by specifying the completeness of information explicitly.

On the other hand, in logic programming, it is typically difficult to open the world. A possible solution is to use the extended logic programming paradigm, where it is possible to explicitly declare an atom to be false using the classical negation sign $\neg$. However, to get the full expressivity of classical negation, it is typically necessary to use disjunctive reasoning. Namely, for each atom $a$, one must assume that either $a$ or $\neg a$ holds. This is a disjunctive statement, and reasoning with it requires more complex algorithms.

The open-world assumption of description logic may seem awkward at first to a user with no experience in formal logic. Namely, relational and object-oriented databases typically employ the closed-world assumption. This does not come as a surprise, since the primary purpose of databases is to efficiently store and retrieve data. Hence, it is intuitive to interpret the negation as meaning ‘not in the database’. On the other hand, description logics have been developed with knowledge representation in mind. Systems for description logic reasoning are not mainly targeted towards storage and retrieval of data, but towards representing knowledge at fine-grained levels of granularity.
There, it makes sense to make a distinction between what the knowledge base knows and what not.

Finally, it is possible to refine the query. Namely, instead of asking ‘does h1 in all worlds only offer only major credit cards’, one might formulate a more precise question: ‘are all credit cards that are known to be offered by h1 known to be major credit cards?’ Such a question can be formulated using the already mentioned autoepistemic operator $\mathbf{K}$ [14], in the following way:

$$\forall \mathbf{K} \text{accepts.} \mathbf{K} \text{MajorCreditCard} \subseteq Q$$

Now h1 definitely is a member of Q, since the only known credit card that h1 offers is Visa, and it is known to be a major credit card.

We finish with a note that the discussion between whether open- or closed-world assumption is more suitable typically ends up in a ‘religious war’ of researchers involved. There are use cases which strongly advocate one, and there are use cases with strongly advocate the other semantics. A versatile system should offer both types of semantics and require allow the user to choose the appropriate one.

4.3 Disjunctive Reasoning

Description logic and logic programming differ significantly in one expressive feature, namely, the disjunction. In logic programming, it is possible only to use disjunction in the body of a rule. For example, in F-Logic/LP variant implemented in Flora, one may use $;$ to denote the disjunction in the body of a rule. Hence, a query for hotels operated by European or US chains might be stated in this way:

$$q(X) :¬ X:hotel[is-part-of \rightarrow C], (C:eu_chain; C:us_chain).$$

However, such usage of disjunction does not really increase the expressivity of the logic. Namely, the above query is equivalent to the union of two non-disjunctive queries:

$$q(X) :¬ X:hotel[is-part-of \rightarrow C], C:eu_chain.$$  
$$q(X) :¬ X:hotel[is-part-of \rightarrow C], C:us_chain.$$  

However, in general first-order logic disjunction can appear in the head of a rule. Technically speaking, such disjunction occurs under the positive polarity. With such features, it is possible to state a disjunctive fact, which is another form of incomplete knowledge: it might be the case that we know that h1 is operated by a European or a US chain, but we are not sure which one. Such incomplete information might be still useful and might allow us deriving useful information. For example, since we know that European and American chains are chains, if we ask for a hotel operated by some chain, we should get h1 as the result. The fact that we do not know which type of chain does h1 belong to does not matter for our query.

In description logics, this situation would be represented by the following knowledge base:

$$\exists is-part-of.(EUC\text{Chain} \cup US\text{Chain})(h1)$$  
$$EUC\text{Chain} \subseteq \text{Chain}$$  
$$US\text{Chain} \subseteq \text{Chain}$$  
$$\exists is-part-of.\text{Chain} \subseteq Q$$
This knowledge base entails $Q(h_1)$, as expected. Reasoning with such a knowledge base can be performed by considering possible different states of the world. In all worlds, $h_1$ is a part of some chain; let us denote this chain with $x$. Now there are two possible worlds for $x$: it may be an EUChain or it may be an USChain. Still, since all instances of EUChain are instances of Chain and all instances of USChain are also instances of Chain, so in both cases $x$ will be an instance of Chain. Hence, in each possible world, $h_1$ is connected to an instance of Chain, so $h_1$ is a part of the answer.

Such reasoning is called reasoning by cases, since it considers all possible cases how incompleteness in the data may be resolved. Reasoning by cases is inherently intractable: assume that there are $n$ individuals for which the membership in one of the two concepts is not specified. For such a knowledge base, there are obviously $2^n$ possible cases which could be explored in the worst case. This gives us an algorithm with an exponential behaviour. To keep such an algorithm tractable, it is necessary to keep the amount of uncertain information to a minimum. However, under this assumption, reasoning by cases is still a powerful knowledge representation feature.

Reasoning by cases is already available in first-order logic. Hence, since F-Logic/logic is an extension of first-order logic, it provides reasoning by cases. However, F-Logic/LP, at least in the Flora and Ontobroker implementations, supports only the Horn subset of first-order logic. Hence, each rule can contain only one literal in the rule head. Each F-Logic/LP program describes exactly one possible world, so reasoning by cases is not possible.

It is worth mentioning that formalisms for logic programming which support reasoning by cases exist. For example, disjunctive datalog [15] is one such formalism. It has been developed primarily with applications such as planning and problem solving in mind. Hence, one may think that principles of disjunctive datalog may be applied to F-Logic/LP and thus obtain a system for disjunctive reasoning in F-Logic. However, we are not certain if this approach is so easy. Namely, the semantics of default inheritance in F-Logic, explained in Subsection 3.3.3, has been defined for the Horn subset of F-Logic. Extending this to include disjunctive reasoning might not be trivial.

4.4 Existential Quantification

Databases (relational and object-oriented) typically allow only asserting information about individuals that can be identified by name. For example, it is possible to represent that the hotel $h_1$ accepts Visa. In this case, the names $h_1$ and Visa are known, and similar information can be represented in, say, a relational database, by a row in the accepts relation, where the first column is the hotel name, and the second column is the credit card type.

However, it might be the case that we know only that $h_1$ accepts some credit card, without knowing exactly which one. In fact, $h_1$ may accept several credit cards, for all we care. The only thing we positively know is that $h_1$ accepts some credit card.

Notice that the name of the accepted card is in this case unknown. Hence, such information cannot be represented easily in traditional database systems: in the above mentioned accepts relation, it is unclear what value to insert into the second column. One possibility is to insert NULL. However, the semantics of NULL is ambiguous: it typically means ‘no value’, but here we use it to denote an ‘unknown value’. To remedy that, in
research many different types of NULL values have been proposed, but these ideas have not found their way into commercial systems yet. Another possibility is to introduce a separate table, where hotels which accept unknown credit cards are appropriately marked. However, this is a hacker’s solution, as it does not truthfully describe the semantics of the intended sentence. Furthermore, it is difficult, if not impossible, to correctly reason about such models.

In first-order logic the above problem is easily solved by using existential quantifiers. Description logic also allows existential quantification, albeit in a slightly restricted way. The above problem might be solved by introducing the following DL axiom:

\[ \exists \text{accepts.CreditCard}(h1) \]

This axiom states that in each possible world, \( h1 \) is connected through \( \text{accepts} \) role to some instance of \( \text{CreditCard} \); we can call this instance \( x \). Such information may be useful in retrieval. For example, the following is a query for all hotels which accept some credit card:

\[ \exists \text{accepts.CreditCard} \subseteq Q \]

Based on the above declarations, one may derive \( Q(h1) \). Allowing to model incompleteness of information obviously improves the expressiveness of the retrieval.

Since F-Logic/logic is an extension of first-order logic, it also provides for existential quantifiers. The above statements might be represented in F-Logic/logic in this way:

\[ \exists C \ h1[\text{accepts} \rightarrow C]. \]
\[ \forall X \ [ (\exists C \ X[\text{accepts} \rightarrow C]) \rightarrow X:q ] \]

In a manner similar to description logic, this knowledge base entails \( h1:Q \). However, for F-Logic/LP it is not quite clear whether and how existential quantifiers are supported. Since Flora is based on the Prolog variant, it does not support existential quantifiers directly. However, the above statement may be represented in F-Logic/LP by means of a well-known transformation for elimination of existential quantifiers called skolemization. This transformation can be described as follows: if the a subformula of the form \( \exists x \varphi(x,y_1, \ldots, y_n) \) occurs under positive polarity in a formula \( \Phi \), then \( \Phi \) is equi-satisfiable with the formula \( \Phi' \), in which the mentioned subformula is replaced with \( \varphi(f(y_1, \ldots, y_n), y_1, \ldots, y_n) \), where \( f \) is a function symbol not occurring in \( \Phi \). The similar holds for a subformula of the form \( \forall x \varphi(x,y_1, \ldots, y_n) \) occurring under negative polarity.

By applying the skolemization principle to the above situation, we get the following F-Logic/LP statements:

\[ h1[\text{accepts} \rightarrow f(h1)]. \]
\[ X:q := X[\text{accepts} \rightarrow C]. \]

For such an F-Logic/LP program, \( h1:q \) is an answer to a query \( ?- X:q \). Although skolemization seems to provide sufficient support for existential quantifiers, in Subsection 4.6 we explain why this is not the case. Briefly, naïve skolemization as described here makes the logic become undecidable. Description logic is, on the other hand, decidable. For a detailed discussion of these issues, please refer to Subsection 4.6.

It is worth noting that Ontobroker version of F-Logic/LP provides support for existential quantifiers. The above statements might be written like this:
EXISTS C h1[accepts -&gt;&gt; C].
X:q := X[accepts -&gt;&gt; C].

However, this is only syntactic sugar: to evaluate such an F-Logic/LP program, Ontobroker applies skolemization, so the discussion from Subsection 4.6 applies.

4.5 Axioms

Support for arbitrary axioms is probably the most obvious difference between F-Logic and description logic. In F-Logic/logic, one can write arbitrary first-order formulae, and in F-Logic/LP one can write arbitrary Horn-type rules. For example, it is possible in F-Logic/logic to define a class of hotels which are located in the same city as the headquarters of the chain that the hotels belong to in the following way:

\[ \forall H : \text{collocated} \leftarrow
\left( \exists R \exists T \exists H : \text{hotel}[\text{located-in} \rightarrow R, \text{is-part-of} \rightarrow C] \land
R : \text{city-area}[\text{is-in} \rightarrow T] \land T : \text{city} \land
C : \text{chain}[\text{headquarters-in} \rightarrow T] \right) \]

Such an axiom can be easily converted into the following F-Logic/LP rule by moving quantifiers to the outermost level:

\[ H : \text{collocated} \leftarrow
H : \text{hotel}[\text{located-in} \rightarrow R, \text{is-part-of} \rightarrow C],
R : \text{city-area}[\text{is-in} \rightarrow T], T : \text{city},
C : \text{chain}[\text{headquarters-in} \rightarrow T] \]

Unfortunately, there is no way to express an equivalent axiom in SHOIQ(D). The reason for this is again decidability. Namely, it is well-known that allowing arbitrary combinations of existential quantifiers and axioms leads to an undecidable logic. For a more detailed discussion of the problems related to decidability, please refer to Subsection 4.6.

Although OWL-DL does not provide for arbitrary axioms, there are logics which do provide such features. A landmark study of the effects of adding free axioms to description logics has been given in [16], where the effects of adding Horn-like rules to various description logics has been studied. The main results of the paper can be summarized as follows:

- Adding free recursive rules to even moderately expressive description logic leads to undecidability.
- Adding free non-recursive rules even to expressive description logic usually does not affect the decidability of the logic.
- If variables in the rules are restricted in a way so that each variable appearing in a role predicate also appears in a predicate not occurring in the description logic knowledge base, then one can use recursive rules without losing decidability.

Another possible approach to integrating description logics and axioms has been presented in [23]. The system presented there is called AL-log, and it requires all description logic predicates in a rule to occur in a non-description logic predicate. In such way, the applicability of rules is restricted only to named individuals, and not to individuals introduced in the model through existential quantification. Although somewhat limited, this is a practical approach which can be very useful in numerous practical situations.
In [17] another proposal for combining rules with OWL-DL has been presented. There, arbitrary rules are allowed and can be freely combined with the description logic. Such an approach automatically leads to an undecidable logic. Another drawback of this approach is that it is unclear how to perform reasoning in such a situation. Namely, the only alternative is to use general first-order theorem proving techniques, such as resolution or basic superposition. Currently considered tableaux approaches cannot be used for such a logic.

4.6 Decidability

Decidability is probably the most controversial feature/requirement of description logic. In particular, the past ten years of description logic research have been mainly driven by this requirement. The primitives available in OWL-DL have been chosen in such a way that the logic remains decidable. In fact, decidability has been the main driving force in creation of OWL-DL. On the other hand, some people believe that decidability is not so important, particularly when it excludes some features which are needed in practice. Because the notion of decidability is so central and controversial in the current state-of-the-art of description logic research, we describe it here in more detail.

4.6.1 What is Decidability?

An algorithm is decidable if:

- given any validly constructed input it always terminates, and
- it always gives a correct answer.

As explained in Subsection 3.1, the main algorithmic problem in logic is to check whether some logical formula is satisfiable or not, i.e. to check if it has a model or not. All other problems, such as query answering or subsumption reasoning can usually be reduced to the problem of satisfiability checking. Hence, if there is a decidable algorithm for satisfiability checking, typically there is a decidable algorithm for other problems. In another words, we need an algorithm that can, for any formula, correctly determine in finite amount of time if the formula is satisfiable or not.

Decidability is a property of an algorithm. However, one colloquially says that a logic is decidable if a decidable algorithm for checking satisfiability of any formula of this logic exists.

It is well-known that first-order logic is semi-decidable. Let $\phi$ be any first-order formula. Algorithms for checking satisfiability of $\phi$ are known, such that they will terminate in finite time if $\phi$ really is unsatisfiable, and they will give an answer ‘yes, $\phi$ is unsatisfiable’. However, it has been shown that, if $\phi$ is an arbitrary satisfiable formula, then an algorithm that terminates for any such $\phi$ does not exist. We would like to make our point clear by stipulating that various algorithms might terminate on various satisfiable formulas $\phi$, but no algorithm can terminate on any formula. Hence, we have ‘one half’ of a decision procedure: we are capable of determining always that $\phi$ is unsatisfiable, but if $\phi$ is satisfiable, we cannot always determine that. That is the meaning of the term ‘semi-decidable’.

4.6.2 Practical Considerations

Undecidability is not welcome in practical applications. Assume that the scenario from Section 2 is realized using an undecidable logic. It would be possible for the user to
create a query for which INT can never give an answer. Note that this is not a bug: the reasoning procedure of INT runs in an infinite loop, but not because it is buggy. There simply cannot be a procedure that will always terminate.

If an undecidable logic is used, a typical solution to this problem is to put a time limit on the time the reasoning procedure can consume to compute an answer. When the time runs out, the procedure simply stops and returns the answers that have been computed up to this point. The main drawback of this approach is that one cannot rely on the results of the reasoning procedure. In our running scenario, if some hotel is not included in the answer list, this does not mean that this hotel does not match the query. Hence, the user must always assume that his query still can be satisfied.

The situations similar to the one described above have been used in the past to motivate the decidability requirement for description logics. However, in practice the above argument can easily be disputed. Namely, most decidable logics still exhibit quite bad worst-case complexity. For example, checking satisfiability of OWL-DL knowledge bases can be performed in time which is in the worst case exponential in the size of the knowledge base with a non-deterministic computational device. This is quite bad from a practical point of view. Hence, it is conceivable that users will issue a query for which computation of all answers will take so long that the fact that the algorithm will eventually terminate does not really matter. The user will probably go to the next provider after a couple of minutes anyway.

Considering the above arguments, we believe that decidability of the logic makes sense. Namely, if the logic is decidable, it is possible to identify the relevant inputs which exhibit the worst-case performance and then apply special optimization techniques. In [18] various optimization techniques have been investigated. It has been shown that acceptable performance of description logics reasoning is achievable.

Another optimization example is presented in [19]: in order to apply optimization strategies from deductive databases to the problem of reasoning in description logic, the authors present a reduction of description logic knowledge bases to disjunctive datalog programs.

If the logic is undecidable, it is difficult to devise optimization strategies for it, since one can never know what one can expect. For example, since disjunctive datalog is decidable, it is theoretically impossible to reduce an undecidable logic to disjunctive datalog. Hence, we believe that it does make sense to consider decidable logics for practical purposes.

4.6.3 Origins of Undecidability

Undecidability arises when an algorithm is required to enumerate an infinite set of objects. Obviously, enumerating such a set would take an infinite amount of time.

In logic, infiniteness is caused by existential quantifiers. Consider, for example, the following statements: “Every person has a parent who is a person”, and “Peter is a person”. From these two statements, it is possible to conclude that Peter has an infinite ancestral line: let $x_1$ be a parent of Peter. But $x_1$ must be a person, so it must have a parent as well; let us call it $x_2$. Obviously, the ancestral line can be extended infinitely.

The above statements can be captured in description logic by the following axioms:
Assume now that someone asks the following query: “Is Peter a grandchild, where a grandchild is defined as someone who has a parent of a parent”. From this natural language description, it is clear that Peter is a grandchild, so the answer should be ‘yes’. In description logic, this is equivalent to checking whether the following knowledge base entails $\text{Grandchild}(\text{Peter})$:

$$\exists \text{parent.} (\exists \text{parent.} \text{Person}) \subseteq \text{Grandchild}$$

Since description logic is decidable, the above question can be answered in finite time. However, before we explain the principles due to which this is possible, we first show why answering such a question is not trivial.

Since the existential quantifier in the rule $\text{Person} \subseteq \exists \text{parent.} \text{Person}$ occurs in the rule head, it must be eliminated using skolemization (see Subsection 4.4). Thus, one becomes the following logic program:

$$\text{parent}(X,f(X)) \ :- \ \text{person}(X).$$
$$\text{person}(f(X)) \ :- \ \text{person}(X).$$
$$\text{grandchild}(X) \ :- \ \text{parent}(X,Y),\text{parent}(Y,Z),\text{person}(Z).$$
$$\text{person}(\text{peter}).$$

Intuitively, the term $f(X)$ should be interpreted like this: for some $X$, $f(X)$ is some parent of it. Obviously, for the symbol $\text{peter}$, one can derive a sequence of parents of the form $f(\text{peter}), f(f(\text{peter}))$, etc.

Now one might try to evaluate this program using a bottom-up (also known as forward-chaining) strategy, to determine the extension of the $\text{grandchild}$ predicate. However, this procedure will not terminate: the set of all grandchildren is obviously infinite. Evaluating a query $\neg \text{grandchild}(\text{peter})$. will terminate in the state-of-the-art rule engine XSB\(^1\), because of the specifics of the top-down (also known as backward-chaining) query evaluation strategy used in XSB. All potential grandchildren might be determined in this way: one might enumerate all constants in a special predicate $c$ (e.g. one would include the assertion $c(\text{peter})$). Now one might try to evaluate the query $\neg \ c(X),\text{grandchild}(X)$. This query will terminate, since it will first match the variable $X$ to a potential constant (e.g. $\text{peter}$), and then check whether the rest of the body holds. However, the query $\neg \ \text{grandchild}(X), c(X)$. would not terminate in XSB, since the subgoal $\text{grandchild}$ would be evaluated before the subgoal $c$.

Evidently, query answering becomes complicated, and termination of the algorithm is dependent on the nuances of how the query is asked. It is worth noting that the solution of appending $c(X)$ would not work in all cases.

The main problem of the approach is skolemization: each model of the skolemized formula looks as presented in Figure 10, part a): for each person, there is an infinite sequence of parents.

However, not all models must look like this. In Figure 10, part b), another model of the knowledge base is presented. In this model, $\text{peter}$ has a parent $x_1$, which has itself as a

\(^1\) http://xsb.sourceforge.net/
parent. This might seem strange: persons cannot have themselves as parents. However, our knowledge base does not state this explicitly. This might be stated by an axiom \( \forall x \neg \text{parent}(x,x) \), which cannot be expressed in description logics. Hence, however strange, Figure 10, part b) is an allowed interpretation for the knowledge base.

Actually, parts a) and b) from Figure 10 have a special relationship. Namely, all nodes \( x_i \) have identical properties to the node \( y_1 \). Under ‘identical properties’, we mean intuitively the following: assume taking a walk from the node peter along the arcs of the model. Then, in either a) of b) part, one would pass through nodes that look the same: each node is an instance of Person and \( \exists \text{parent}. \text{Person} \) concepts, and each node in a) is connected with exactly one node which is similar to the corresponding node in b). Hence, there is no way to determine that \( x_i \) and \( y_1 \) are different, if one does not look at the node name, but only looks at node’s properties. A more technical term is to say that each \( x_i \) is bisimilar with \( y_1 \).

The essence of decidable reasoning in description logic is to consider only finite models which are bisimilar to any infinite model. We refrain from presenting the actual algorithm, but only mention that the technique of blocking is crucial for decidable reasoning [20]. Research in reasoning in description logics focuses on creating various blocking techniques, suitable for various classes of logic. It is also worth noting that blocking is not the only technique: among others, various automata techniques [21] or resolution-based techniques [22] can be used to obtain a decidable reasoning procedure for description logics.

However, none of the named techniques are typically part of logic programming environments. Hence, mere application of skolemization does not yield decidable reasoning procedures for various types of logics. Integrating such techniques with logic programming does not seem to be trivial.

Mentioned techniques, such as blocking, can be applied only if the underlying logic is decidable. Under the presence of existential quantifiers, as discussed in Subsection 4.5, allowing general axioms leads to undecidability. That is the reason why description logics do not allow general axioms. Also, it should be clear by now that logic programming with general axioms and function symbols is not decidable. This also applies to F-Logic/LP.

![Figure 10. Bisimulation-equivalent Models](image-url)
However, if existential quantifiers occurring under positive polarity and function symbols are not allowed, a logic becomes decidable even with general axioms. In this case, there are no implied constants, so there is no infinite universe that one should examine when answering a query.

This restriction can be relaxed even further. Namely, logic programming with existential quantifiers and function symbols is decidable if one does not allow recursive rules. In this case, the logic program can be evaluated bottom-up, but there is still no danger of generating functional terms of infinite depth. Hence, logic programming, and F-Logic/LP in particular, can be made decidable by employing these restrictions.

Currently there is no consensus which features are generally needed. We believe that, whereas decidability is certainly important for all applications, the importance of existential quantifiers or cyclic definitions is dependent on the application, so it is not possible to give a general statement on which one should take precedence.

4.7 Logic-based vs. Conceptual Modelling Approach

This section briefly surveys advantages and disadvantages of using a conceptual modelling and database approach, versus using a (description) logic approach, for the description and management of ontologies.

4.7.1 Data Modelling

Modelling constructs (e.g. object types) in EER conceptual models support direct modelling of rich data structures, leading to representations that are close to how we perceive things in the real world. In particular, this close matching between representations and real world things leads to synthetic schemas that are easily apprehended. Instead, most description logics rely on simple binary data structures. Representing the real world with such simple constructs leads to an explosion in the number of axioms that are needed, similarly to what happens with relational databases (where representation of a single real world entity may spread over many tables). For example, a one-page EER schema would require several pages of description logics axioms to describe the same representation. In addition, having only binary structures blurs the distinction between what describes composite things (e.g. entities and links) and what merely describes a property. One could say that the reader of a DL description has to perform a reverse engineering process to reconstruct something that resembles her/his perception of the real world that is described. EER conciseness is definitely an advantage of conceptual models, in particular when thinking of schemas designed by humans or shown to humans (but a computer agent would also have an easier task in exploring a conceptual schema than in exploring a long list of DL axioms). The advantage extends to visualization of the ontology structure, where again EER diagrams are likely to be easier to capture at a glance than the DL diagrams supported by recent DL editors.

In terms of supporting description of derived concepts, the advantage currently goes to DL. Some conceptual models support a few derived concepts (e.g. derived object types, derived classes, derived attributes), whose instances and values can be automatically inferred. But they do not support concepts that designers would define by a logical formula without knowing where they will fit in the generalization hierarchy or even knowing the generalization hierarchy. This is possible in DL, where users can precisely define new concepts by a logical formula as complex as needed.
Thanks to the power of their logic reasoners, DL approaches allow constraints to also be defined by logic formulae (of type inclusion, equivalence, and disjointedness). The inference mechanisms automatically check the consistency of the new definitions and constraints, deduce where the new constructs are placed in the generalization hierarchy, and infer their instances.

Conceptual models have a number of predefined integrity constraints (e.g. cardinality constraints, key constraints) that are very easily described (more easily than in DL). However, to support a declarative formulation of general integrity constraints, they have to resort to an associated logic language (FOL is usually sufficient).

4.7.2 Data Manipulation

Instance creation is unconstrained in DL. Instances may be created without being attached to a concept. The creation of a new instance may not conform to the rules described by the axioms. In fact, the creation of an instance leads to one of three cases: 1) the instance fully conforms to existing axioms; 2) the instance contradicts existing axioms, in which case the user is warned about the contradiction; and 3) the instance neither fully conforms nor contradicts existing axioms, in which case the DL reasoner infers that there currently is some missing knowledge that, if known, would make the new instance conforming to the axioms. Indeed, description logic systems naturally adhere to the open world assumption, which assumes that present data is just the explicitly known subset of the valid data, and more valid data may be inferred by sophisticated reasoning. For example, if axioms state that every hotel has a name, the creation of a new hotel is accepted even if no name is attached to the new hotel.

On the contrary, databases follow the closed world assumption, stating that only information that is present in the database (or derivable by explicitly defined derivation rules) is valid. If a fact is not in the database, the fact is considered false. As a consequence, creation of new instances has to obey all integrity constraints that apply to the instance. For example, if the schema prescribes that every hotel instance must hold a value for the hotel name, the creation of a new hotel without specifying its name is not accepted.

It is not easy to evaluate which approach is better. In fact, each one is best suited for the purpose it has been designed for. DL and its open world assumption fit well within an environment where the ontology is incrementally defined, which corresponds to a situation such that at each stage the current ontology only holds part of the world of interest, hence there are many more specifications that are relevant but not yet entered into the ontology. They also fit well with the idea that ontologies evolve as a result of collaborative design, where many independent partners can contribute new specifications to the ontology. In this case a strong consistency checking mechanism is needed (e.g. satisfiability checking).

The database approach and its closed world assumption fit well in normative environments, where the ontology has to interact with an information system which assumes that the data it uses comes in a given format and is consistent with the application rules that have been stated in the ontology. Consequently, database systems simply do not need sophisticated reasoners to infer additional information.

In terms of querying the ontology and its instances, databases and description logics offer complementary functionality for instance querying. Databases systems usually provide powerful assertional query languages, complemented with efficient query opti-
mization tools. Description logic systems support a set of simple functions for accessing instances and derived facts computed by their inference engines. Simply stated, the difference is that databases have been purposely designed to store and manage huge volumes of data instances, while DL approaches have typically been targeted at sophisticated reasoning over a relatively small volume of instances.

### 4.7.3 Constraints

Checking the consistency of the set of constraints and checking the consistency between the constraints and the schema are tasks that can be performed automatically by the reasoners available in description logics.

On the other hand, databases have a normative approach. It is not possible to define a schema that does not obey the meta-model constraints, and it is not possible to enter data that do not conform to the schema. Satisfiability issues have little impact (and are usually discarded). Decidability issues do not arise as database languages do not allow expressing queries that would not terminate. Instead, databases have no or little reasoning facilities, they cannot check the set of integrity constraints. Moreover, usually they do not even have an integrated language for defining integrity constraints.

### 4.7.4 Beyond Data Structures

Part of the semantics of the real world comes from where things are located in space and time. Traditional modelling approaches (in DL as in conceptual modelling approaches) ignore these components, assuming that the real world of interest is now and here. On the contrary, there is a huge number of applications where spatio-temporal information is essential. Considering our hotel example, spatial information could be used to convey the actual geographical location of hotels, cities and countries. This would enable queries such as “find hotels within 10 miles of a given city”. Similarly, room rates are a typical example of information that is valid only within a given time period. In the current description, this is captured using the attributes fromDate and toDate. However, this is a poor solution in the sense that only the user is aware of the temporal semantics of these attributes. From the system viewpoint, these are two ‘normal’ attributes, with a Date domain. No temporal reasoning and no temporal operators (in the sense developed by research in temporal databases) will be deployed by the system on such data.

There has been quite an investment in the DL community to develop temporal extensions of DL languages. There have been only few efforts to similarly develop spatial extensions. Spatio-temporal DLs are a research item for the future.

The picture is comparatively better in conceptual modelling, where several proposals for spatio-temporal conceptual models exist today, and there is a pretty good understanding of what are the required functionalities.

### 4.7.5 Hybrid Paradigm

In conclusion, conceptual modelling and the database approach provide better responses to the requirements in terms of readability and ease of understanding of the ontology description, and in terms of efficiency in ontology management (storing and querying large ontologies, i.e. ontologies with large volume of instances). DL approaches naturally provide better responses in terms of declarative description of generic integrity constraints, and in terms of reasoning over the axioms and inferring new knowledge from the explicitly defined knowledge. All of these requirements coexist in modern web-based and interoperable systems. It is worth raising the question whether a hybrid
system, where the database component and the logic component would cooperate, each one performing the tasks for which it is best suited, might be the most promising solution for semantically rich information management, in particular semantic web information services.
5 Practical Requirement: Hybrid Reasoning

The discussion in Section 4 clearly shows that various types of logic are available, each tailored to different purposes. These logics exhibit quite different features, the choice of which is guided by certain intellectual and practical principles. Currently, there is no clear winner and no logic can be unconditionally considered preferred to others.

In fact, it is not even possible to achieve consensus at the level of individual features. Consider, for example, open- vs. closed-world semantics: in some applications, open worlds may be preferred, whereas in applications more similar in functionality to classical database applications, closed world semantics is more natural. Actually, in most applications, a combination of the two might in fact be preferred. Thus, the user can choose the primitive which suits him best and can even apply them simultaneously.

The problems facing the IT industry in near future are quite complex, as the use case sketched in Section 2 testifies. Hence, we believe that such complex problems will require complex problem solving techniques. For problems which can benefit from automated reasoning techniques, the main question determining the success or the failure of the solution will not be which logic, but how to combine existing logical paradigms. Combining features of different logics in a coherent reasoning framework is what we call hybrid reasoning, and it is the main topic of WP1 of the DIP project.

In the rest of this section we give the requirements on a framework for hybrid reasoning. These requirements are formulated as a list of features/logical constructs, as well as a specification of how these hybrid features might interact.

It is important to understand that this document presents the requirements, without any guarantee that they can be met. It might happen that in the course of the project, it becomes clear that some requirements cannot be fulfilled.

5.1 Supported Formalisms

The framework should enable interoperability between most currently considered formalisms:

- description logics,
- (disjunctive) logic programming,
- F-Logic/LP,
- relational databases.

As commonly known, F-Logic/LP can be encoded in common Horn logic programming. Hence, the main focus is on integrating description logics and logic programming; integration with F-Logic/LP follows immediately.

5.2 Level of Interoperability

None of the formalisms should be significantly reduced in order to achieve interoperability. Rather, we propose to carefully design interfaces between the formalisms. These interfaces might be restricted to achieve other goals, such as decidability.

Interoperability between formalisms should be two-way, i.e. each component of the hybrid reasoning framework should be able to contribute with its inferences to other
components. In this way we hope to avoid rigid layering of logic components, where one component is strictly subordinate to the other component.

5.3 Decidability
Based on considerations from Subsection 4.6 and on a general consensus of project partners, decidability is a clear requirement on any logical framework. Although the framework might offer undecidable primitives, it should be possible to perform reasoning in known decidable fragments using known decidable algorithms.

5.4 Open- and Closed-World Assumption
Instead of forcing the user to choose open or closed world semantics for the entire logical system, a hybrid reasoning framework should support both open and closed world reasoning. Ideally, the user should be able to precisely control each axiom and state whether it is to be evaluated under open- or closed-world assumption. Similarly, it should be possible to express open- and closed-world queries.

A promising mechanism for achieving this is the already mentioned autoepistemic operator $K$ [14]. However, the exact relationship between the semantics of $K$ and existing non-monotonic semantics, such as well-founded or stable model semantics is not clear. Clarifying these relationships might be the starting point for addressing this requirement.

5.5 Support for Non-monotonic Negation
The framework should support non-monotonic negation. Ideally, this should be well-founded negation. This requirement is mainly needed to support F-Logic/LP. However, other forms of negation may be considered.

As elaborated in Subsection 4.2, the semantics of the negation is closely related to open- and closed-world assumption. Hence, this requirement is interdependent with the requirement 5.4.

5.6 Support for Arbitrary Axioms
The framework for hybrid reasoning should support asserting arbitrary axioms. Combining axioms with description logic should be decidable. In order to design an expressive, but a decidable solution, results from [16] and [23] should be considered.

Since F-Logic/LP already allows writing axioms leading to undecidability (e.g. recursive axioms with function symbols), it should be possible to state and use such axioms, in order to respect the requirement 5.2.

5.7 Acceptable Performance Level
In order to achieve a practicable solution, it is crucial to pay attention to the performance of the framework. Since the formalisms under consideration are quite heterogeneous, unless special attention is paid, the hybrid reasoning framework runs the risk of being just a research prototype. To make this requirement more concrete, we set our goals in the following way: the system implemented in WP1 should provide responses within a couple of seconds for ontologies consisting of $10^5$ concepts and $10^6$ individuals.
Although the solution to this problem is open, we believe that techniques from deductive database research, such as magic sets [24], are key to fulfilling this requirement.

5.8 Follow the Principle of ‘Graceful Degradation’

Logical formalisms being considered vary significantly in the computational complexity and the performance of practical reasoning systems. Ideally, reasoning should be performed in an ‘as good as possible’ manner. Hence, if some ‘expensive’ features of some formalism, such as disjunctive reasoning, are not used, there should not be a performance penalty. In this way, the performance of the system degrades gracefully, as one uses more expressive features.
6 CONCLUSION

In this document, we present the state-of-the-art in various frameworks for knowledge representation. In particular, we focus on description logic and F-Logic, two formalisms which are currently considered as standards for ontology reasoning in the Semantic Web. We motivate the need for a logical formalism by a practical usage scenario from data integration. Such scenarios are becoming increasingly common, as the level of B2B interoperability increases.

Apart from merely presenting existing formalisms, we contrast in detail the features present in each of them. In this way, we point out the commonalities, but also the differences between the formalisms.

We show that there is no clear consensus on the set of features required to solve complex problems, such as the presented use-case scenario. Furthermore, different features of various formalisms do not exclude, but often complement each other. Ideally, a hybrid reasoning framework is needed, which would integrate features of each of the formalism. In such a way, the user is allowed to select and combine the features to solve his problem at hand.

We derived a set of requirements that such a framework should support. Our main goal is to support each formalism as-is, without reducing its expressivity. Rather, we propose to create controlled reduced interfaces between formalisms. Furthermore, the fact that one can combine various formalisms should not influence the performance of reasoning. The framework should exhibit graceful degradation of performance: if ‘expensive’ modelling primitives are not used, the performance of the system should be as in the case one works with a formalism where this primitive is not in present at all.

It is important to understand that requirements presented in this document are just requirements. Although some intuitive ideas for realising these goals exist, it may turn out that some goals cannot be met in practice.
REFERENCES


