Proceedings of the 2001 Workshop on Applications of Description Logics

Editors: Günther Görz, Volker Haarslev, Carsten Lutz, and Ralf Möller

Vienna, Austria
September 18, 2001

Abstract

Recently, a growing interest in description logics and their applications can be observed. This is mainly due to the development of very expressive description logics and optimized description logic systems which support terminological and/or assertional reasoning for these logics. This workshop intended to gather researchers as well as practitioners who are interested in description logics and their applications. The primary focus of this workshop was on applications of description logics. Ian Horrocks, University of Manchester, gives a tutorial-style talk about latest developments in description logic research.

These proceedings can also be found at http://www.CEUR-WS.org.

Technical papers

Reasoning on UML Class Diagrams using Description Logic Based Systems

Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
lastname@dis.uniroma1.it

Abstract

In this paper we study how automated reasoning systems based on Description Logics (DLs) can be used for reasoning about UML class diagrams. The ability of reasoning automatically on UML class diagrams makes it possible to provide computer aided support during the application design phase in order to automatically detect relevant properties, such as inconsistencies and redundancies. We show that UML class diagrams can be formalized as knowledge bases expressed in the DL $\mathcal{DLR}$. $\mathcal{DLR}$ knowledge bases can be translated into knowledge bases expressed in the variants of $\mathcal{ALCQI}$ accepted by state-of-the-art DL-based systems. Hence, in principle, the reasoning capabilities of such systems can be used to reason on UML class diagrams. However, we report some experiments indicating that state-of-the-art systems have still difficulty in dealing with the resulting knowledge bases.

1 Introduction

The *Unified Modeling Language* (UML) is the de facto standard formalism for object-oriented modeling [1, 11]. There is a vast consensus on the need for a precise semantics for UML [9, 14], in particular for UML class diagrams. Indeed, several kinds of formalizations of UML class diagrams have been proposed in the literature [8, 9, 10, 7]. Many of them have been proved very useful with respect to the task of establishing a common understanding of the formal meaning of UML constructs. However, to the best of our knowledge, none of them has the explicit goal of building a solid basis for allowing automated reasoning techniques, based on algorithms that are sound and complete wrt the semantics, to be applicable to UML class diagrams.

We are interested in exploiting the research on Description Logics (DLs), which are decidable logics tailored towards class based knowledge representation, to carry out various forms of reasoning on UML class diagrams, so as to provide support during the specification phase of software development. Recently the research on DLs has resulted in a number of automated reasoning systems [15, 16, 17, 12, 13], that have been successfully tested in various application domains (see e.g., [19, 20, 18]). Such systems are candidates to form the core reasoning engine for advanced UML CASE tools.

In this paper, we illustrate a formalization of UML class diagrams in terms of DLs [2]. In particular, we show how UML class diagrams can be captured by knowledge bases expressed in the DL $\mathcal{DLR}$ [4, 3]. This logic is particularly well tailored towards the high expressiveness of UML information structuring mechanisms, and allows one to easily model important additional properties, such as disjointness of classes, or partitions of classes into subclasses, that are typically specified by means of constraints in UML class diagrams. $\mathcal{DLR}$ assertions can be translated into $\mathcal{ALCQI}$ assertions. Since variants of the latter are accepted by state-of-the-art DL-based reasoning systems, in principle, we can exploit such systems to reason about UML class diagrams. However, in spite of the fact that such systems have shown to perform nicely in several context, we report in this paper some experiments indicating that they still have serious efficiency problems when dealing with UML class diagrams.

The rest of the paper is organized as follows. In Section 2 we give a brief overview of the Description Logic $\mathcal{DLR}$. In Section 3 we show how UML class diagrams can be formalized in $\mathcal{DLR}$. In Section 4 we discuss the use of DL-based reasoning systems, namely FaCT [17] and Racer [13], for reasoning about UML class diagrams, and show some results of our experimentation with such systems. Section 5 concludes the paper. In the appendix, we show the UML class diagrams used in the reported experiments.

2 The Description Logic $\mathcal{DLR}$

The basic elements of $\mathcal{DLR}$ [4, 3] are *concepts* and *n-ary relations*. We assume to deal with a finite set of atomic relations and atomic concepts, denoted by P and A, respectively. Arbitrary relations (of given arity between 2 and n_{max}), denoted by R, and arbitrary concepts, denoted by C, are built according to the following syntax:

$$\begin{array}{lcl} R & ::= & \top_n \mid P \mid (i/n : C) \mid \neg R \mid R_1 \sqcap R_2 \\ C & ::= & \top_1 \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid (\leq k\,[i]R) \end{array}$$

where i denotes a component of a relation, i.e., an integer between 1 and n_{max}, n denotes the *arity* of a relation, i.e., an integer between 2 and n_{max}, and k denotes

$$\begin{array}{rcl@{\qquad}rcl}
\top_n^{\mathcal{I}} & \subseteq & (\Delta^{\mathcal{I}})^n & \top_1^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\
P^{\mathcal{I}} & \subseteq & \top_n^{\mathcal{I}} & A^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \\
(i/n : C)^{\mathcal{I}} & = & \{t \in \top_n^{\mathcal{I}} \mid t[i] \in C^{\mathcal{I}}\} & (\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\neg R)^{\mathcal{I}} & = & \top_n^{\mathcal{I}} \setminus R^{\mathcal{I}} & (C_1 \sqcap C_2)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\
(R_1 \sqcap R_2)^{\mathcal{I}} & = & R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}} & (\leq k\,[i]R)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \sharp\{t \in R_1^{\mathcal{I}} \mid t[i] = a\} \leq k\}
\end{array}$$

Figure 1: Semantic rules for $\mathcal{DLR}$ (P, R, R_1, and R_2 have arity n)

a non-negative integer. We consider only concepts and relations that are *well-typed*, which means that (i) only relations of the same arity n are combined to form expressions of type $R_1 \sqcap R_2$ (which inherit the arity n), and (ii) $i \leq n$ whenever i denotes a component of a relation of arity n.

We also make use of the following abbreviations: $C_1 \sqcup C_2$ for $\neg(\neg C_1 \sqcap \neg C_2)$, $C_1 \Rightarrow C_2$ for $\neg C_1 \sqcup C_2$, $(\geq k\,[i]R)$ for $\neg(\leq k{-}1\,[i]R)$, $\exists[i]R$ for $(\geq 1\,[i]R)$, $\forall[i]R$ for $\neg\exists[i]\neg R$. Moreover, we abbreviate $(i/n : C)$ with $(i : C)$, when n is clear from the context.

A $\mathcal{DLR}$ *knowledge base* (KB) is constituted by a finite set of *inclusion assertions*, where each assertion has one of the forms:

$$R_1 \sqsubseteq R_2 \qquad\qquad C_1 \sqsubseteq C_2$$

with R_1 and R_2 of the same arity.[1]

The semantics of $\mathcal{DLR}$ is specified through the notion of interpretation. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of a $\mathcal{DLR}$ KB $\mathcal{K}$ is constituted by an *interpretation domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and to each relation R of arity n a subset $R^{\mathcal{I}}$ of $(\Delta^{\mathcal{I}})^n$, such that the conditions in Figure 1 are satisfied (in the figure, $t[i]$ denotes the i-th component of tuple t). We observe that $\top_1$ denotes the interpretation domain, while $\top_n$, for $n > 1$, does *not* denote the n-Cartesian product of the domain, but only a subset of it that covers all relations of arity n. It follows, from this property, that the "$\neg$" constructor on relations is used to express difference of relations, rather than complement.

To specify the semantics of a KB we first define when an interpretation satisfies an assertion as follows: An interpretation $\mathcal{I}$ *satisfies* an inclusion assertion $R_1 \sqsubseteq R_2$ (resp. $C_1 \sqsubseteq C_2$) if $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$ (resp. $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$). An interpretation that satisfies all assertions in a KB $\mathcal{K}$ is called a *model* of $\mathcal{K}$.

Several reasoning services are applicable to $\mathcal{DLR}$ KBs. The most important ones are KB satisfiability and logical implication. A KB $\mathcal{K}$ is *satisfiable* if there

[1] $\mathcal{DLR}$ knowledge bases may also include *identification-constraints* that allow one to force instances of concepts or relations to be uniquely identified through suitable mechanisms (see [4] for details). Interestingly, however, such additional constraints play no role in checking knowledge base satisfiability or logical implication of inclusion assertions. For this reason in this paper we do not consider them.

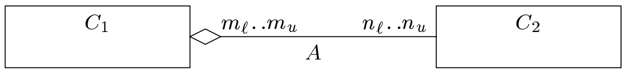


Figure 2: Aggregation in UML

exists a model of $\mathcal{K}$. An inclusion assertion α is *logically implied* by $\mathcal{K}$ if all models of $\mathcal{K}$ satisfy α. One can easily verify that logical implication and KB (un)satisfiability are mutually reducible.

One of the distinguishing features of $\mathcal{DLR}$ is that it is equipped with reasoning algorithms that are sound and complete wrt to the semantics. Such algorithms allow one to decide all the above reasoning tasks in deterministic exponential time [4, 3].

3 Representing UML class diagrams

We concentrate on UML class diagrams for the conceptual perspective. Hence, we do not deal with those features that are relevant for the implementation perspective, such as public, protected, and private qualifiers for methods and attributes.

Classes A *class* in an UML class diagram denotes a *set of objects* with common features, hence it can be represented by a $\mathcal{DLR}$ concept. This follows naturally from the fact that both UML classes and $\mathcal{DLR}$ concepts denote *sets of objects*. Attributes and operations of classes can be easily represented by means of $\mathcal{DLR}$-relations [2].

Relationships between classes come in two forms in UML: aggregations, denoting part-whole relationships, and associations, denoting general relationships between two or more classes.

Aggregations An *aggregation* in UML, graphically rendered as in Figure 2, is a binary relation between the instances of two classes, denoting a generic form of part-whole relationship, i.e., a relationship that specifies that each instance of a class is made up of a set of instances of another class. An aggregation A, saying that instances of the class C_1 have components that are instances of the class C_2, is formalized in $\mathcal{DLR}$ by means of a binary relation A together with the following assertion:

$$A \sqsubseteq (1\!:\!C_1) \sqcap (2\!:\!C_2).$$

Note that the distinction between the contained class and the containing class is not lost. Indeed, we simply use the following convention: *the first argument of the relation is the containing class.* The multiplicity of an aggregation can be

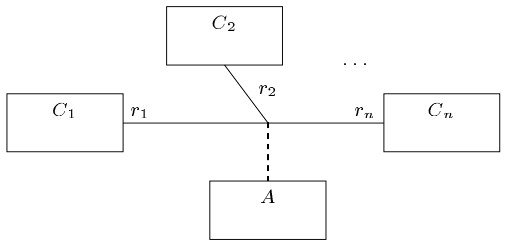


Figure 3: Association in UML

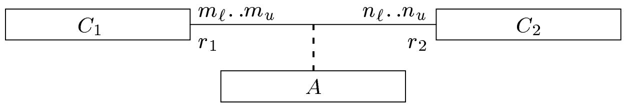


Figure 4: Binary association in UML

easily expressed in $\mathcal{DLR}$. For example, the multiplicities shown in Figure 2 are formalized by means of the assertions:

$$\begin{array}{rcl} C_1 & \sqsubseteq & (\geq n_\ell\,[1]A) \sqcap (\leq n_u\,[1]A) \\ C_2 & \sqsubseteq & (\geq m_\ell\,[2]A) \sqcap (\leq m_u\,[2]A) \end{array}$$

Associations An *association* in UML, graphically rendered as in Figure 3, is a relation between the instances of two or more classes. An association often has a related *association class* that describes properties of the association such as attributes, operations, etc.

Since associations have often a related association class, we formalize associations in $\mathcal{DLR}$ by reifying each association A into a $\mathcal{DLR}$ concept A with suitable properties. We represent an association among n classes $C_1, \dots, C_n$, as shown in Figure 3, by introducing a concept A and n *binary* relations $r_1, \dots, r_n$, one for each component of the association A. Each binary relation r_i has C_i as its first component and A as its second component. Then we introduce the following assertion:

$$\begin{array}{rcl} A & \sqsubseteq & \exists[1]r_1 \sqcap (\leq 1\,[1]r_1) \sqcap \forall[1](r_1 \Rightarrow (2\!:\!C_1)) \sqcap \\ & & \qquad\vdots \\ & & \exists[1]r_n \sqcap (\leq 1\,[1]r_n) \sqcap \forall[1](r_n \Rightarrow (2\!:\!C_n)) \end{array}$$

where $\exists[1]r_i$, with $i \in \{1, \dots, n\}$, specifies that the concept A must have all components $r_1, \dots, r_n$ of the association A, $(\leq 1\,[1]r_i)$ specifies that each such component is single-valued, and $\forall[1](r_i \Rightarrow (2\!:\!C_i))$ specifies the class each component has to belong to.[2]

[2]In addition, we would need an identification constraint saying that the relations $r_1, \dots, r_n$

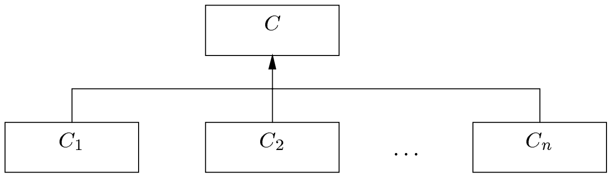


Figure 5: A class hierarchy in UML

For a binary UML association, we can easily represent multiplicities by imposing suitable number restrictions on the $\mathcal{DLR}$ relations modeling the components of the association. The multiplicities shown in Figure 4 are captured as follows:

$$\begin{array}{rcl} C_1 & \sqsubseteq & (\geq n_\ell\,[1](r_1 \sqcap (2\!:\!A))) \sqcap (\leq n_u\,[1](r_1 \sqcap (2\!:\!A))) \\ C_2 & \sqsubseteq & (\geq m_\ell\,[1](r_2 \sqcap (2\!:\!A))) \sqcap (\leq m_u\,[1](r_2 \sqcap (2\!:\!A))) \end{array}$$

Generalization In UML one can use *generalization* between a parent class and a child class to specify that each instance of the child class is also an instance of the parent class. Hence, the instances of the child class inherit the properties of the parent class, but typically they satisfy additional properties that do not hold for the parent class.

Generalization is naturally supported in $\mathcal{DLR}$. If an UML class C_2 generalizes a class C_1, we can express this by the $\mathcal{DLR}$ assertion:

$$C_1 \sqsubseteq C_2$$

Inheritance between $\mathcal{DLR}$ concepts works exactly as inheritance between UML classes. This is an obvious consequence of the semantics of inclusion assertions, which is based on subsetting. Indeed, in $\mathcal{DLR}$, given an assertion $C_1 \sqsubseteq C_2$, every tuple in a relation having C_2 as i-th argument type may have as i-th component an instance of C_1, which is in fact also an instance of C_2. As a consequence, in the formalization, each attribute or operation of C_2, and each aggregation and association involving C_2 is correctly inherited by C_1. Observe that the formalization in $\mathcal{DLR}$ also captures directly inheritance among association classes, which are treated exactly as all other classes, and multiple inheritance between classes (including association classes).

In UML, one can group several generalizations, as shown e.g.., in Figure 5, and impose covering or mutual disjointness between classes, if needed. This is captured in $\mathcal{DLR}$ by a set of inclusion assertions, one between each child class

form an identifier of the concept A. However, as mentioned, such a constraint has no impact on reasoning on logical implication or satisfiability of the resulting knowledge base, so we omit it here.

and the parent class:

$$C_i \sqsubseteq C \qquad \text{for each } i \in \{1, \ldots, n\}$$

Then if the superclass C is a *covering* of the subclasses $C_1, \ldots, C_n$, we include the additional assertion

$$C \sqsubseteq C_1 \sqcup \cdots \sqcup C_n$$

For each pair of subclasses C_i and C_j that are *mutually disjoint*, we include the assertions

$$C_i \sqsubseteq \neg C_j$$

Constraints In UML it is possible to add information to a class diagram by using *constraints*. In general, constraints are used to express in an informal way information which cannot be expressed by other constructs of UML class diagrams. One can exploit the expressive power of $\mathcal{DLR}$ to formalize several types of constraints that allow one to better represent the application semantics and that are typically not dealt with in a formal way. This allows one to take such constraints fully into account when reasoning on the class diagram.

4 Experiments

We have formalized as $\mathcal{DLR}$ knowledge bases several UML class diagrams. Then we have used state-of-the-art DL-based systems to reason with them, by translating the $\mathcal{DLR}$ knowledge bases into $\mathcal{ALCQI}$ knowledge bases (or more precisely knowledge bases expressed in the variants of $\mathcal{ALCQI}$ accepted by the systems used). In particular, we have used the two systems FaCT [3] (the executable $\mathcal{SHIQ}$ reasoner (shiq-app.exe) contained in the Corba-FaCT distribution v.2.15, excluding the corba interface) and Racer [4] (v.1-5-10). We have run the experiments on a Pentium III biprocessor, 866 Mhz, 512MB of RAM and OS Windows 2000 Professional.

Below we report the results obtained with four rather simple UML class diagrams, shown in the appendix: Restaurant, Library, Soccer, and Hospital, modeling, respectively, a restaurant, a library, a soccer championship and the acceptance procedure in a hospital. The reasoning service we focused on is satisfiability of the class diagram. Observe that all diagrams are obviously satisfiable. The obtained results are shown in Table 1, where:

- complete refers to the original UML class diagrams;

[3]Available at `http://www.cs.man.ac.uk/~horrocks/FaCT`.
[4]Available at `http://kogs-www.informatik.uni-hamburg.de/~race`.

	Restaurant		Hospital		Soccer		Library	
	FaCT	Racer	FaCT	Racer	FaCT	Racer	FaCT	Racer
no mult. constr.	yes	yes	yes	yes	yes	yes	yes	yes
no minimal mult. constr.	yes	no	yes	yes	yes	yes	yes	yes
no maximal mult. constr.	yes	no	yes	no	yes	no	yes	yes
complete	no	no	yes	no	no	no	yes	no

Table 1: Successful classification of the considered UML class diagrams

- no multiplicity const. refers to the class diagrams weakened by removing all multiplicity constraints, i.e., making all multiplicities of the form $0..*$;
- no minimal multiplicity const. refers to the class diagrams weakened by removing minimal multiplicity constraints, thus getting multiplicities of the form $0..*$ or $0..1$;
- no maximal multiplicity const. refers to the class diagrams weakened by removing maximal multiplicity constraints, thus getting multiplicities of the form $0..*$ or $1..*$.

In the table, "yes" indicates that the reasoner could classify the knowledge base corresponding to the UML class diagram, and "no" that the reasoner couldn't classify it because it ran out of resources.

When the reasoners are able to classify a knowledge base (yes in the table), they both take less than 1 minute to perform the classification. When FaCT cannot classify a knowledge base (no in the table), this is because it goes in stack overflow (in about 1 minute on the experiments reported). Observe that the only limit to the stack size is the one imposed by the OS, and FaCT goes in stack overflow whenever the OS can't provide more memory. FaCT memory requests increase quite regularly, until all the available memory is exhausted. As for Racer, when it cannot classify a knowledge base (no in the table, this is because it starts paging, so that all the resources are used to perform memory swaps and the CPU usage decreases greatly. After 1 hour of paging we stopped the reasoner. The only exception to this behaviour is in classifying the knowledge base corresponding to Hospital with no maximal multiplicity constraints, where Racer goes in stack overflow, even setting the stack size to the maximum.

FaCT can classify all knowledge bases corresponding to the class diagrams having no minimal multiplicity constraints and those having no maximal multiplicity constraints, but it can't classify some of those corresponding to the complete class diagrams: namely Soccer and Restaurant, which are characterized by having cycles of associations/aggregations all involving minimal multiplicity constraints in both directions. [5]

[5] Curiously, we noticed that FaCT is able to classify the knowledge base corresponding

RACER can classify none of the knowledge bases corresponding to the complete UML class diagrams. Instead, it can classify the knowledge bases corresponding to the weakened class diagrams with no multiplicity constraints, and those corresponding to the class diagrams with no minimal multiplicity constraints, with the exception of Restaurant. The weakened class diagrams with no maximal multiplicity constrains are too complex for the current version of RACER, with the exception of Library, where only few minimal multiplicity constraints appear.

From an analysis of the UML class diagrams and the corresponding knowledge bases, it appears that what makes reasoning difficult for the current systems is the combination of: (1) terminological cycles involving existentials (which in UML class diagrams are generated by minimal multiplicity constraints); (2) inverse roles (which are intrinsic in the possibility of navigating UML aggregations and associations components in both directions); (3) functional restrictions combined with existential restrictions (which are present in the complete class diagrams); (4) the overall size of the UML class diagrams.

More information about the conducted experiments, including the $\mathcal{DLR}$ knowledge bases corresponding to the UML class diagrams considered here, and the knowledge bases expressed in the languages accepted by FACT and RACER, are available at `http://www.dis.uniroma1.it/~berardi/uml2dl`.

5 Conclusions

We have seen that UML class diagrams can be formalized as DL knowledge bases, and this potentially allows for exploiting DL-based reasoning systems to perform various kinds of reasoning on them. However, the experimentation with state-of-the-art DL reasoners, shows that the current reasoners may have serious efficiency problems in dealing with the resulting knowledge bases. Observe that all results obtained apply also to Entity-Relationship diagrams (with cardinality constraints) [6, 5], which are tightly related to UML class diagrams.

Hence, we encourage further research on practical DL reasoners. Reasoning with UML class diagrams (with multiplicity constraints) can be a challenging testbed for them.

Acknowledgments We would like to thank the developers of FACT and RACER, and in particular Ian Horrocks, Sergio Tessaris, Volker Haarslev, and

to Restaurant, if we reverse the direction of two aggregations (related and is_comprised), which in this case amounts to reversing the order of the two arguments of the $\mathcal{DLR}$ relation corresponding to two aggregations. This appears quite strange, considering that $\mathcal{DLR}$ relations are reified in $\mathcal{ALCQI}$ and the treatment of the two components in the translation of the relations is completely symmetrical.

Ralf Möller, for their kind and very helpful assistance during the experimentation with their systems.

References

[1] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide.* Addison Wesley Publ. Co., Reading, Massachussetts, 1998.

[2] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Reasoning on UML class diagrams in description logics. In *Proc. of IJCAR Workshop on Precise Modelling and Deduction for Object-oriented Software Development (PMD 2001)*, 2001.

[3] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of PODS'98*, pages 149–158, 1998.

[4] D. Calvanese, G. De Giacomo, and M. Lenzerini. Identification constraints and functional dependencies in description logics. In *Proc. of IJCAI 2001*, 2001. To appear.

[5] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Use of the reconciliation tool at Telecom Italia. Technical Report DWQ-UNIROMA-007, DWQ Consortium, Oct. 1999.

[6] D. Calvanese, M. Lenzerini, and D. Nardi. Unifying class-based representation formalisms. *J. of Artificial Intelligence Research*, 11:199–240, 1999.

[7] T. Clark and A. S. Evans. Foundations of the Unified Modeling Language. In D. Duke and A. Evans, editors, *Proc. of the 2nd Northern Formal Methods Workshop.* Springer-Verlag, 1997.

[8] A. Evans, R. France, K. Lano, and B. Rumpe. The UML as a formal modeling notation. In H. Kilov, B. Rumpe, and I. Simmonds, editors, *Proc. of the OOPSLA'97 Workshop on Object-oriented Behavioral Semantics*, pages 75–81. Technische Universität München, TUM-I9737, 1997.

[9] A. Evans, R. France, K. Lano, and B. Rumpe. Meta-modelling semantics of UML. In H. Kilov, editor, *Behavioural Specifications for Businesses and Systems*, chapter 2. Kluwer Academic Publisher, 1999.

[10] A. S. Evans. Reasoning with UML class diagrams. In *Second IEEE Workshop on Industrial Strength Formal Specification Techniques (WIFT'98).* IEEE Computer Society Press, 1998.

[11] M. Fowler and K. Scott. *UML Distilled – Applying the Standard Object Modeling Laguage.* Addison Wesley Publ. Co., Reading, Massachussetts, 1997.

[12] V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of IJCAI 2001*, 2001.

[13] V. Haarslev and R. Möller. RACER system description. In *Proc. of IJCAR 2001*, 2001.

[14] D. Harel and B. Rumpe. Modeling languages: Syntax, semantics and all that stuff. Technical Report MCS00-16, The Weizmann Institute of Science, Rehovot, Israel, 2000.

[15] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of KR'98*, pages 636–647, 1998.

[16] I. Horrocks and P. F. Patel-Schneider. Optimizing description logic subsumption. *J. of Log. and Comp.*, 9(3):267–293, 1999.

[17] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of LPAR'99*, number 1705 in LNAI, pages 161–180. Springer-Verlag, 1999.

[18] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments*, pages 85–91, 1995.

[19] D. McGuinness and J. Wright. Conceptual modelling for configuration: A description logic-based approach. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing Journal*, 12:333–344, 1998.

[20] U. Sattler. *Terminological Knowledge Representation Systems in a Process Engineering Application.* PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, 1998.

A Appendix

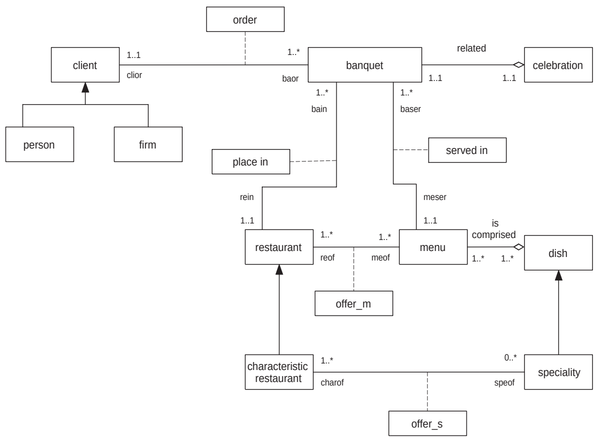


Figure 6: UML class diagram: RESTAURANT

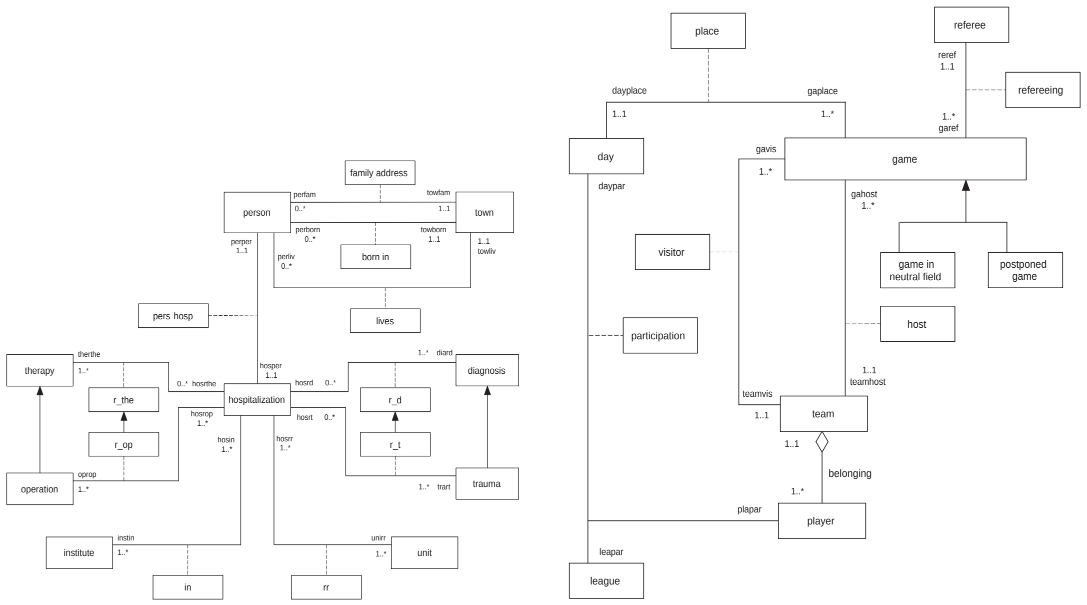


Figure 7: UML class diagrams: Hospital and Soccer

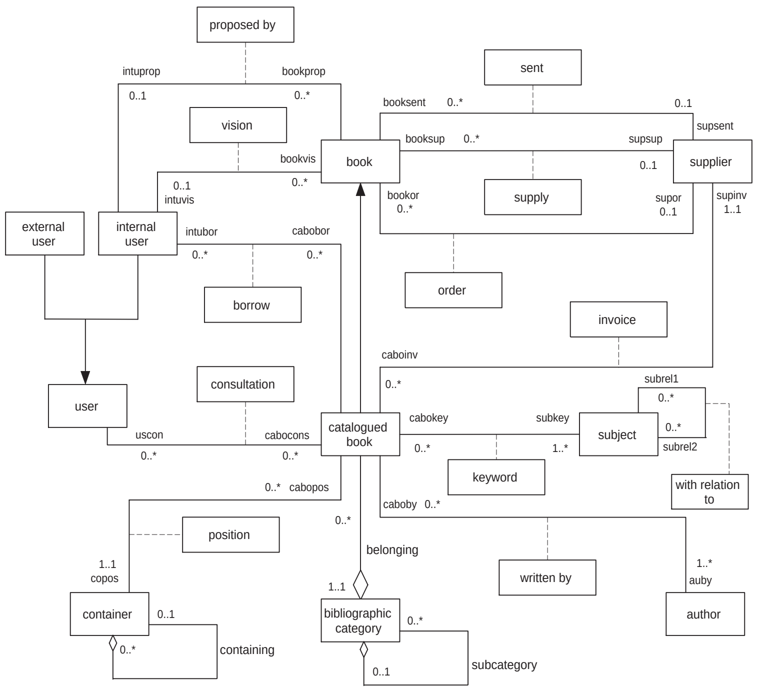


Figure 8: UML class diagram: Library

Using Non-standard Inferences in Description Logics—what does it buy me? *

Sebastian Brandt and Anni-Yasmin Turhan,
Theoretical Computer Science,
RWTH Aachen, Germany
Email: {sbrandt, turhan}@cs.rwth-aachen.de

Abstract

In knowledge representation systems based on Description Logics, standard inference services such as consistency, subsumption, and instance are well-investigated. In contrast, non-standard inferences like most specific concept, least common subsumer, unification, and matching are missing in most systems—or exist only as ad-hoc implementations. We give an example of how these inferences can be applied successfully in the domain of process engineering. The benefit gained in our example, however, occurs in to many domains where knowledge bases are managed by persons with little expertise in knowledge engineering.

1 Process Engineering

As an application domain for knowledge representation systems based on Description Logics (DL-systems) in general, and certain non-standard inferences in particular, we give a brief introduction to the basic notions of the field of process engineering. In this context, a *process* is defined as a sequence of physical, chemical, biological, and informational operations intentionally executed to change substances in respect to their nature, properties, and composition.

Process engineering is concerned with methods, tools, and their management for the design and control of a process.[1] Here, *models* are used to represent, analyze, and optimize processes and get a deeper understanding of their nature. In general, a model is an abstraction of some object under consideration characterized by a lower level of complexity while retaining some of the original properties of interest.

*This work has been supported by the DFG, Project BA 1122/4-1.

[1] *Plant engineering* in turn deals with the actual (chemical) plant performing the process and its construction, which is abstracted from in process engineering.

In process engineering, exact equation-based mathematical models are particularly desirable because of their high predictive capabilities in numerical analysis and simulation. Unfortunately, even for simple chemical processes, such models are too complex for ad-hoc construction by hand. Nevertheless, adequate models can be obtained step by step, starting with other representation formalisms, e.g., so-called *block-oriented* models. In such models, a process is represented by an undirected graph with blocks as vertices and connections as edges. Each *block* stands for a standardized sub-unit of the entire process with certain interfaces and each *connection* for a flow of material, energy, or information. The type of a connection linking two interfaces of blocks is determined by the interface specifications. Typically, block-oriented modeling environments have a *block repository* in which building blocks are stored.

During the life-cycle of a chemical process, several models on different levels of detail are involved. In an early design stage, for instance, rather crude models allow to consider alternative designs in minimal time. Once one of them proved promising, more accurate models are used for further examination. With such a cascade of models, however, it is not clear how to benefit from one modeling stage when going into further detail on the next.

In answer to this, several requirements have been identified for block-oriented models and appropriate modelling environments in process engineering [15]:

- *Variable granularity*: The model should allow composite building blocks, i.e., blocks again comprising blocks and connections. These can be decomposed during the design phase until the desired level of detail is reached.

- *Generic building blocks*: A block in the repository should not be fully specified but rather represent a class of some subunit. During the design procedure, particular instances are obtained by specifying the relevant variables, equations, and values abstracted from in the classes.

- *Structured storage*: To avoid unnecessary extensions of the block repository and to facilitate browsing and searching, the existing blocks should be arranged in an "is-specialization-of" hierarchy.

- *Automatic classification*: If the specialization order would be derivable automatically, the system could additionally maintain consistency of new building blocks during the design procedure and locate the correct positions for their storage in the repository.

- *Re-use of submodels*: It should be possible to store (abstractions of) subunits in existing models in the repository for later re-use.

- *Maintenance support*: As the block repository typically will be developed over a long period of time by many people, detecting redundancies and integrating additional repositories should be possible.

The challenge to meet these requirements has inspired a cooperation between the Institute for Process Systems Engineering at RWTH Aachen, where a prototype modelling environment is being developed, and our research group, where DL-systems are studied. It has already been shown that DL-systems can successfully be employed for most of the above tasks [15]. Testing the developed prototype environment has provided additional insights. When designing models by means of block-oriented modelling environments, process engineers showed two characteristic strategies for the design of new (generic) blocks:

- *Bottom-up design*: From several existing process models, the process engineer selects a certain collection of subunits deployed for a similar purpose. She then introduces a new generic block as an abstraction of these units.
- *Design by modification*: Before assembling a new generic block from scratch, the knowledge engineer tries to locate a structurally similar one in the repository. She then modifies the existing block to suit the new requirements.

In this work, we will show how these design strategies can be supported by non-standard inferences offered by DL-systems. In the following section, we will introduce Description Logics formally and discuss their benefit for the requirements mentioned above. Sections 3 and 4 describe the particular non-standard inference services used to support the two design techniques.

2 Description Logics and Process Modelling

Description Logics (DL) form a category of knowledge representation (KR) formalisms used to represent terminological knowledge in a structured and well-defined way. A DL-system consists of a *knowledge base* together with certain *inference services*. The knowledge base comprises two components, the *TBox* and the *ABox*. Intuitively, the TBox defines the vocabulary by which a concrete world (in this application a process model) is described in the ABox. Both are defined by means of *concepts*, whose syntax and semantics is introduced next.

Concepts are inductively defined using a set of concept *constructors*, starting from a set N_C of *concept names* and a set N_R of *role names*. The constructs available in the DLs considered here are listed in Table 1. In $\mathcal{EL}$, the top-concept ($\top$), conjunction ($C \sqcap D$), and existential restriction ($\exists r.C$) are allowed. $\mathcal{ALE}$ additionally provides the bottom-concept ($\bot$) and primitive negation ($\neg A$). $\mathcal{ALN}$ extends $\mathcal{ALE}$ with number restrictions ($\geq n\, r$) and ($\leq n\, r$), but does not provide existential restrictions. A concept defined over the DL $\mathcal{L}$ ($\mathcal{L} \in \{\mathcal{EL}, \mathcal{ALE}, \mathcal{ALN}\}$) is referred to as $\mathcal{L}$-concept.

The semantics of concepts is defined in terms of an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. The *domain* $\Delta^{\mathcal{I}}$ is a non-empty set, and the *interpretation function* $\cdot^{\mathcal{I}}$

Syntax	Semantics	$\mathcal{EL}$	$\mathcal{ALE}$	$\mathcal{ALN}$
$\top$	$\Delta^{\mathcal{I}}$	x	x	x
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	x	x	x
$\forall r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y\colon (x,y) \in r^{\mathcal{I}} \to y \in C^{\mathcal{I}}\}$		x	x
$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y\colon (x,y) \in r^{\mathcal{I}} \to y \in C^{\mathcal{I}}\}$	x	x	
$\bot$	$\emptyset$		x	x
$\neg A$, $A \in N_C$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$		x	x
$(\geq n\, r)$, $n \in \mathbb{N}$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}}\} \geq n\}$			x
$(\leq n\, r)$, $n \in \mathbb{N}$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}}\} \leq n\}$			x

Table 1: Syntax and semantics of concepts.

maps every concept name $A \in N_C$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each role name $r \in N_R$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The second column of Table 1 shows how $\cdot^{\mathcal{I}}$ is extended to complex concepts.

Definition 1 (TBox) *A TBox $\mathcal{T}$ is a finite set of* concept definitions *of the form $A \doteq C$, where $A \in N_C$ and C is a concept. Every concept name A may occur only once on a left-hand side in $\mathcal{T}$. If it does, then A is called* defined, *otherwise* primitive. *In DLs providing primitive negation only primitive concepts may be negated on the right-hand side of concept definitions. An interpretation $\mathcal{I}$ is a* model *for $\mathcal{T}$ iff $A^{\mathcal{I}} = C^{\mathcal{I}}$ for every $A \doteq C \in \mathcal{T}$.*

To illustrate the introduced notions of concept, concept definition and TBox, consider an example TBox.

Example 2 *The $\mathcal{ALE}$-TBox $\mathcal{T}_{ex}$ contains the following concept definitions inspired by the process engineering domain:*

$$\begin{array}{rcl}
\mathsf{Liquid} & \doteq & \neg\mathsf{Solid} \sqcap \neg\mathsf{Gas}, \\
\mathsf{Container} & \doteq & \mathsf{Volume} \sqcap (\forall\,\mathsf{contains.\,Substance}), \\
\mathsf{FluidTank} & \doteq & \mathsf{Container} \sqcap (\forall\,\mathsf{hasConnection.\,Port}) \sqcap \\
 & & (\exists\,\mathsf{contains.\,Liquid}), \\
\mathsf{Pipeline} & \doteq & \mathsf{Volume} \sqcap \mathsf{Tube} \\
 & & (\forall\,\mathsf{hasConnection.}\,(\mathsf{Port} \sqcap (\exists\,\mathsf{hasPart.\,Valve}))) \sqcap \\
 & & (\forall\,\mathsf{contains.\,Substance}) \sqcap (\exists\,\mathsf{contains.}\neg\mathsf{Solid})
\end{array}$$

In the TBox $\mathcal{T}_{ex}$ the concept Liquid *is defined as something that is no* Gas *and no* Solid. *A* Container *is defined as a* Volume *containing only* Substances. *Based on these two defined concepts, a* FluidTank *is defined as a* Container *which contains a* Liquid *and is only connected to* Ports. *Finally, a* Pipeline *is defined as a* Volume *and a* Tube *and is only connected to* Ports *which in turn must have a* Valve *as a part. Furthermore, a* Pipeline *must contain something, which is no* Solid *and all it contains are* Substances.

To represent knowledge about an actual instance of the application domain, individuals and their interrelations are described in an ABox. Thus, in addition to N_C and N_R, we introduce a finite set N_I of *individual names.* Formally, an ABox can now be defined as follows:

Definition 3 (ABox) *An ABox $\mathcal{A}$ is a finite set of* concept assertions *of the form* $\mathsf{a}\colon C$ *and* role assertions *of the form* $(\mathsf{b},\mathsf{c})\colon r$*, where* $\mathsf{a},\mathsf{b},\mathsf{c} \in N_I$*, C is an arbitrary concept, and $r \in N_R$ a role name. An interpretation $\mathcal{I}$ is a* model *for $\mathcal{A}$, iff* $\mathsf{a}^{\mathcal{I}} \in C^{\mathcal{I}}$ *and* $(\mathsf{b}^{\mathcal{I}},\mathsf{c}^{\mathcal{I}}) \in r^{\mathcal{I}}$ *for every* $\mathsf{a}\colon C$ *and every* $(\mathsf{b},\mathsf{c})\colon r$ *in $\mathcal{A}$. For every interpretation $\mathcal{I}$, every* $\mathsf{a} \in N_I$ *is mapped to some* $\mathsf{a}^{\mathcal{I}} \in \Delta^{\mathcal{I}}$*, such that* $\mathsf{a} \neq b$ *implies* $\mathsf{a}^{\mathcal{I}} \neq b^{\mathcal{I}}$ *(unique name assumption).*

In our process engineering application, each of the individual blocks is represented by an individual in an ABox. The generic blocks from the repository are represented by concepts defined in a TBox. Thus, TBox and ABox form a knowledge base for all blocks constructed in the modelling environment as illustrated in Figure 1.

To derive implicit knowledge from the explicit one given in the knowledge base, there are three so-called *standard inferences*, namely *consistency*, *subsumption*, and *instance*, as defined below.

Definition 4 (Standard inferences) *A concept C is* consistent *iff there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$. A concept C is* subsumed *by a concept D (written $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all interpretations $\mathcal{I}$. The concepts C and D are* equivalent *(written $C \equiv D$) iff they subsume each other. An individual name $a \in N_I$ is an* instance *of C w.r.t. an ABox $\mathcal{A}$ and its TBox $\mathcal{T}$ (written* $\mathsf{a} \in_{\mathcal{A},\mathcal{T}} C$*) iff* $\mathsf{a}^{\mathcal{I}} \in C^{\mathcal{I}}$ *for every model $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$.*

These inferences are essential for almost all DL-systems. Especially, computing the so-called subsumption hierarchy of concepts yields the "is-specialization-of"-hierarchy mentioned in Section 1. Algorithms deciding subsumption form the basis for structured storage and the algorithms for computing the "instance-of"-relation realize the automatic classification of objects. Not all of the tasks mentioned in Section 1 can be accomplished by means of standard inferences, e.g., they do not facilitate the previously mentioned design strategies utilized by process engineers. This is where the *non-standard inferences* come into play.

3 Supporting the Bottom-up Approach

The bottom-up generation of a new block (i.e., concept) from a set of process models (i.e., ABox individuals) selected by the domain expert is realized by non-standard inferences in two steps. Firstly, the most specific concept is computed for each of the selected ABox individuals, such that the individual is an instance

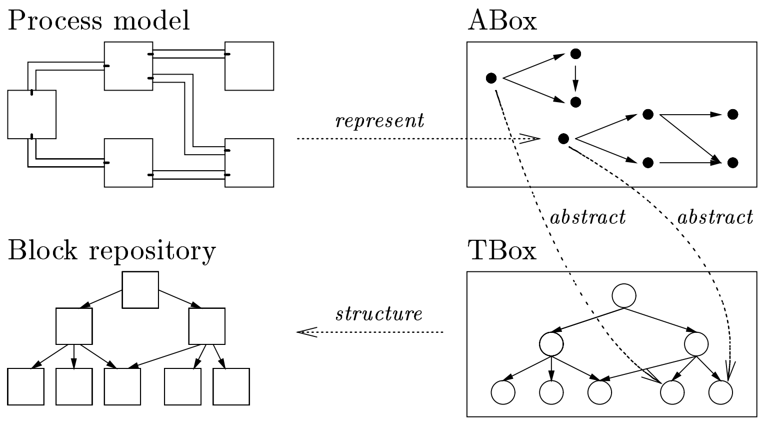


Figure 1: Modelling environment and knowledge base

of the obtained concept which is most specific w.r.t. subsumption. Next, a single concept is computed from all the obtained concepts, which subsumes all the obtained concepts and is also the most specific concept to do so. The resulting concept is then offered to the process engineer for inspection and further processing and, if suitable, added to the generic block repository. The first step is realized by the non-standard inference *most specific concept* (msc) which is defined in the following way:

Definition 5 (msc) *Let $\mathcal{A}$ be an $\mathcal{L}$-ABox,* a *an individual in $\mathcal{A}$ and C a concept in $\mathcal{L}$, then C is the* most specific concept (msc) *of* a *w.r.t. $\mathcal{A}$ ($\mathsf{msc}_{\mathcal{A}}(\mathsf{a})$) iff* $\mathsf{a} \in_{\mathcal{A}} C$*, and for all $\mathcal{L}$-concepts C',* $\mathsf{a} \in_{\mathcal{A}} C'$ *implies $C \sqsubseteq C'$.*

Computing the msc of an individual yields an abstraction from a concrete individual and from its interrelationships expressed in the ABox by generalizing it into a concept.

Example 6 *As an example inspired from the application domain, we want to describe a distillation device which takes sea-water as an input and separates it into water and salt. Such a device could be represented by an $\mathcal{EL}$-ABox $\mathcal{A}_{ex}$ with the following assertions:*

$$\begin{array}{rlrl}
\mathsf{device}: & \mathsf{MarineDistiller}, & \mathsf{seawater}: & \mathsf{solution} \sqcap \mathsf{Liquid}, \\
(\mathsf{device},\mathsf{seawater}): & \mathsf{hasInput}, & (\mathsf{seawater},\mathsf{water}): & \mathsf{contains}, \\
(\mathsf{device},\mathsf{water}): & \mathsf{hasOutput}, & (\mathsf{seawater},\mathsf{salt}): & \mathsf{contains}, \\
(\mathsf{device},\mathsf{salt}): & \mathsf{hasOutput}, & \mathsf{water}: & \mathsf{Solvent} \sqcap \mathsf{Liquid}, \\
 & & \mathsf{salt}: & \mathsf{Solute} \sqcap \mathsf{Solid}
\end{array}$$

Note that the individuals water *and* salt *occur as role-successors for both of the individuals* device *and* seawater*. The* msc(device) *w.r.t. the underlying ABox $\mathcal{A}_{ex}$ is now given by:*

$$\begin{aligned}\mathsf{msc}_{\mathcal{A}_{ex}}(\mathsf{device}) = \; & \mathsf{MarineDistiller} \sqcap \\ & \exists\, \mathsf{hasInput}.\, (\mathsf{Solution} \sqcap \mathsf{Liquid} \sqcap \\ & \qquad\qquad \exists\, \mathsf{contains}.\, (\mathsf{Solvent} \sqcap \mathsf{Liquid}) \sqcap \\ & \qquad\qquad \exists\, \mathsf{contains}.\, (\mathsf{Solute} \sqcap \mathsf{Solid})) \sqcap \\ & \exists\, \mathsf{hasOutput}.\, (\mathsf{Solvent} \sqcap \mathsf{Liquid}) \sqcap \\ & \exists\, \mathsf{hasOutput}.\, (\mathsf{Solute} \sqcap \mathsf{Solid})\end{aligned}$$

In the obtained concept the concept names from the ABox $\mathcal{A}_{ex}$ are preserved in the msc *concept and the interrelations are expressed by existential restrictions. The co-references in $\mathcal{A}_{ex}$ to each of* water *and to* salt *can not be captured in the concept, instead the concepts from $\mathcal{A}_{ex}$ corresponding to these individuals are duplicated.*

Unfortunately, the msc need not always exist due to cyclic relationships between ABox individuals such as $\{(\mathsf{a},\mathsf{b})\colon r,\ (\mathsf{b},\mathsf{a})\colon r\} \subseteq \mathcal{A}$. An individual from a cyclic ABox may be instance of all concepts from an infinite sequence of concepts C_1, C_2, ... where each concept C_i encodes one more traversal of the cycle expressed in the ABox than C_{i-1} and is thereby more specific than all its predecessors in the infinite sequence of concepts. Since the individual is an instance of *all* C_i s, the most-specific concept would be $\sqcap_{i=1}^{\infty} C_i$, which cannot be expressed in every DL with existential restrictions. For further details, refer to [2].

However, for DLs providing existential restrictions the msc for cyclic ABoxes can be approximated by the so-called *k-approximation.* The k-approximation is a msc whose nesting depth of quantifiers is bounded by k ($k \in \mathbb{N}$). See [12] for details. Once in our process engineering application the k-approximation or, if possible, the msc of each selected individual block is attained, the subsuming concept—the least common subsumer—of them is computed. It is defined as follows:

Definition 7 (lcs) *Let $\mathcal{T}$ be an $\mathcal{L}$-concept and C_1, ..., C_n concepts in $\mathcal{L}$ from $\mathcal{T}$, then C is the* least common subsumer (lcs) *of C_1, ..., C_n w.r.t. $\mathcal{T}$ ($\mathsf{lcs}_{\mathcal{T}}(C_1, \ldots, C_n)$) iff $C_i \sqsubseteq_{\mathcal{T}} C$ for all $1 \leq i \leq n$, and for all $\mathcal{L}$-concepts C', $C_i \sqsubseteq_{\mathcal{T}} C'$ for all $1 \leq i \leq n$ implies $C \sqsubseteq C'$.*

Thus, both the msc and the lcs generalize the input yielding the most specific concept w.r.t. the underlying TBox; only that the msc refers to a single ABox individual while the lcs refers to several concepts based on conccepts defined in a TBox.

Example 8 *Let us consider the* lcs *of the concepts* FluidTank *and* Pipeline *as defined in the TBox $\mathcal{T}_{ex}$ from Example 2. First, both concepts have to be unfolded w.r.t. $\mathcal{T}_{ex}$, i.e., all names of defined conepts are replaced recursively by the right-hand sides of their concept definitions. Next, the* lcs *concept of both input concepts is computed. In this case we obtain:*

$$\begin{aligned}\mathsf{lcs}_{\mathcal{T}_{ex}}\,(\mathsf{FluidTank}, \mathsf{Pipeline}) =\ & \\ & \mathsf{Volume} \sqcap (\forall\, \mathsf{hasConnection.\, Port}) \sqcap \\ & (\forall\, \mathsf{contains.\, Substance}) \sqcap (\exists\, \mathsf{contains.}\, (\mathsf{Substance} \sqcap \neg\, \mathsf{Solid})).\end{aligned}$$

The lcs *concept reflects that both input concepts are a* Volume*, because* Volume *lies in the intersection of the concept names on top-level of both input concepts. For the concept occurring in value restrictions, the* lcs *algorithm is applied recursively for each role. The existential restrictions in the* lcs *concept are obtained by conjoining the concepts in the existential restriction and those in their corresponding value restriction for each of the input concepts and then recursively applying the* lcs *algorithm to the obtained conjunctions.*

For the DLs introduced in Section 2, the lcs always exists. The lcs (as well as the msc, if it exists) is uniquely determined up to equivalence. In our research group, algorithms for the lcs have been developed for several DLs, see [10, 5, 2, 11].

Equipped with the non-standard inferences msc and the lcs, the demand from our application domain to construct knowledge bases in a bottom-up fashion can be met. The knowledge engineer selects some fully specified blocks that should form the new generic block for the repository. Then the blocks are automatically translated into individuals in an ABox, representing the parts and properties of each of the blocks. Next, the DL-system computes the msc of each of them and then generalizes them into a single concept by computing the lcs. The resulting concept is then translated back into the representation used in the modelling environment and offered as a new block to the process engineer, see Figure 1. Note that the domain expert is not involved in the "DL-part" of this process, therefore our method is suitable for users with little KR expertise.

In our application, the lcs inference does not only support the bottom-up approach for augmenting the repository. It may in addition be employed to obtain a well-structured storage in the repository which in turn is necessary for easily retrieving generic blocks for a possible re-use. So, if a generic block in the repository has many specializations, say $B_1, \ldots, B_n$ for a large number n, and the process engineer searches for a building block to re-use, inspecting all of the B_is to find a candidate may not be practical. New generic blocks subsuming some of the B_is and thereby providing an intermediate level in the specialization hierarchy facilitates browsing it. Such intermediate blocks can be derived by computing the least common subsumers of some of the B_is and adding them to the repository.

The lcs w.r.t. TBoxes has been implemented for the DL $\mathcal{ALE}$. As seen in Example 8, all input concepts have to be unfolded completely against the underlying TBox before computing the actual lcs concept. It is well-known that unfolding a concept can cause an exponential blow-up of the concept size [14].

Therefore, the concepts to be handled and—even worse—returned by the lcs algorithm can become very large. This does not only slow down the computation of the lcs, but also yields unreadable concepts. First empirical evaluations of our lcs-implementation applied to TBoxes from the process engineering domain have shown that the returned concepts fill several pages of output and are therefore too big to be readable and comprehensible for a human reader, see [7, 16].

Thus, for assessment by domain experts, the resulting concepts have to be represented more compactly. To this end, our research group investigates methods for finding a *minimal rewriting* of a concept w.r.t. the underlying TBox [6]. In a minimal rewriting, parts of the concept are replaced by names defined in the TBox. The effect of such a rewriting is somewhat inverse to unfolding, e.g., in Example 8 the lcs can be represented in a more compact way by using the definition in $\mathcal{T}_{ex}$ and replacing the sub-term of the lcs concept "$\mathsf{Volume} \sqcap (\forall \mathsf{contains.Substance})$" by "$\mathsf{Container}$".

For DLs with existential restrictions, the computations of a minimal rewriting involves a high degree of non-determinism. Therefore, we have to resort to heuristics yielding small but not always minimal rewritings. In the case of the TBox used in our process engineering application, for instance, employing such heuristics to concepts of size 800 yields concepts of size 10. Refer to [7, 16] for details.

Moreover, rewritings can be used to “translate” concepts from one DL $\mathcal{L}_1$ into concepts from another, less expressive DL $\mathcal{L}_2$ by computing the best *approximation* of the concept. This service is especially desirable if more inference services are available in $\mathcal{L}_2$.

4 Supporting the Modification Approach

Another useful non-standard inference is matching, which was first proposed in the DL-system CLASSIC [13, 9]. In order to define matching, we need to introduce *concept patterns.*

Let $\mathcal{L}$ be any of the DLs introduced in Section 2 together with the sets N_C, N_R, and N_I. Additionally, let N_X be a finite set of *concept variables* disjoint to $N_C \cup N_R \cup N_I$. *$\mathcal{L}$-concept patterns* are $\mathcal{L}$-concepts for which in addition concept variables can be used in the place of concept names—except for the fact that the primitive negation ($\neg$) may not occur in front of variables. A *substitution* σ is a mapping from N_X into the set of $\mathcal{L}$-concepts. It is extended to concept patterns P by replacing every occurrence of $X \in N_X$ in P by $\sigma(X)$. Thus, $\sigma(P)$ again is an $\mathcal{L}$-concept. With these preliminaries we can define matching problems as follows:

Definition 9 (matching problems) *An* $\mathcal{L}$-matching problem *is of the form $C \equiv^? P$, where C is an $\mathcal{L}$-concept and P an $\mathcal{L}$-concept pattern. A substitution*

σ is a matcher *for $C \equiv^? P$ iff $C \equiv \sigma(P)$, i.e., σ replaces the variables in P by concepts in such a way that equivalence holds.*

As a trivial example, consider the matching problem $\mathsf{A} \sqcap \forall \mathsf{r}.\mathsf{B} \equiv^? X \sqcap \forall \mathsf{r}.Y$. An appropriate matcher would be, for instance, $\{X \mapsto \mathsf{A} \sqcap \forall \mathsf{r}.\mathsf{B}, Y \mapsto \mathsf{B}\}$.

Intuitively, if a concept can be matched against a pattern P, then their syntax trees share the "upper part", i.e., where P is fully specified, while deviations may occur at leaves labelled with variables. Hence, the set of all concepts that can be matched against P contains infinitely many concepts structurally similar to P to some extent. In this sense, matching P against several concepts and returning those which can be matched, can be seen as a search with the fully specified part of P as search criterion.

Let us return to the process engineer designing a new generic block in a block-oriented modelling environment as described in Section 1. For the strategy of design by modification, the crucial step is to find a generic block in the repository *structurally* similar to what the knowledge engineer intends to design. As an example, assume a modelling task for a fluid tank equipped with a cooling system and an equivalent backup cooling system, each with a thermostat controller. A convenient starting point for the design could be a block comprising a fluid tank combined with an arbitrary device controlled by some control units and a similar backup device.

Example 10 *With a matching algorithm at hand, relevant generic blocks could be found by matching the following concept pattern against every concept in the KB.*

$$\begin{aligned}\exists \mathsf{hasPart}.(\mathsf{FluidTank} \sqcap \exists \mathsf{hasPart}.(\neg \mathsf{BackupDevice} \sqcap X \sqcap \exists \mathsf{hasPart}.\mathsf{Controller})\\ \sqcap \exists \mathsf{hasPart}.(\mathsf{BackupDevice} \sqcap X \sqcap \exists \mathsf{hasPart}.\mathsf{Controller}))\end{aligned}$$

The pattern specifies blocks consisting of at least one fluid tank equipped with at least two equivalent components one of which is a backup device and one not. Both components must have a controller.

The query could thus also return blocks with two or more tanks or, for instance, with duplicate heating systems or stirring devices. Nevertheless, by additionally retrieving a block representing a single cooling device and another for a single thermostat control unit, the engineer is well prepared to complete the design task efficiently. Naturally, usability issues suggest to hide the formal construction of patterns by user-friendly query front-ends.

Note that in all admissible concepts, both occurrences of X must be replaced by the same concept. This structural constraint cannot be expressed by simple "wildcards" familiar from ordinary search engines.

Formal means exist for further refinement of such pattern-driven searches. For the DL $\mathcal{ALN}$, so-called *side conditions* have been proposed to restrict the concepts a variable may be replaced with [1]. In the above example, we may want to restrict the query to duplicate devices with, say, temperature-related functionality. To this end, we could use a side condition of the form

$$X \sqsubseteq^? \mathsf{ThermalDevice},$$

thus including only those devices represented by the concept ThermalDevice.

Apart from supporting the technique of design by modification, matching can also help to provide maintenance support for the block repository, as described in Section 1 [8]. Matching algorithms for deciding and solving matching problems have been proposed in the DLs $\mathcal{ALE}$ and $\mathcal{ALN}$ [3, 4]. In $\mathcal{ALE}$, the decision problem is NP-complete, whereas the computation problem is EXPTIME-complete. In $\mathcal{ALN}$, the decision and computation problem are polynomial.

5 Conclusion and Future Work

We have presented an application where non-standard inference services can significantly enhance the usability of DL-systems. Here these services were proposed to assist process engineers in their practical techniques of designing process models. These techniques, however, are not specific to this very domain but apply to any scenario where knowledge bases are managed by domain experts with little expertise in knowledge representation.

With the DLs presented here, not all properties of the described models can be represented sufficiently. The demand for more expressive DLs, however, also necessitates to adapt the existing inference services to new language constructors such as qualified number restrictions, transitive roles, and role hierarchies.

Approaches to capture the relevant extensions by appropriate algorithms for non-standard inferences are currently studied by our research group. Additional language constructors can further increase the computational complexity of such algorithms. Nevertheless, experience has shown that a high worst-case complexity often can be tolerated as long as a moderate average-case complexity is observed in practical applications.

A promising alternative might be to realize non-standard inferences for expressive description logics by approximating the input concepts in a less expressive DL, where the desired inferences can be realized more efficiently.

References

[1] F. Baader, S. Brandt, and R. Küsters. Matching under side conditions in description logics In B. Nebel, editor, *Proc. of IJCAI-01*, 2001.

[2] F. Baader and R. Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In O. Herzog and A. Günter, editors, *KI-98*, volume 1504 of *Lecture Notes in Computer Science*, pages 129–140, Bremen, Germany, 1998. Springer-Verlag.

[3] F. Baader and R. Küsters. Matching in description logics with existential restrictions. In *Proc. of KR2000*, pp. 261–272, Morgan Kaufmann Publishers, 2000.

[4] F. Baader, R. Küsters, A. Borgida, and D. McGuinness. Matching in description logics. *Journal of Logic and Computation*, 9(3):411–447, 1999.

[5] F. Baader, R. Küsters, and R. Molitor. Computing least common subsumer in description logics with existential restrictions. In T. Dean, editor, *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99)*, pages 96–101, Stockholm, Sweden, 1999. Morgan Kaufmann, Los Altos.

[6] F. Baader, R. Küsters, and R. Molitor. Rewriting concepts using terminologies. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-00)*, pages 297–308, San Francisco, CA, 2000. Morgan Kaufmann Publishers.

[7] F. Baader and R. Molitor. Building and structuring description logic knowledge bases using least common subsumers and concept analysis. In B. Ganter and G. Mineau, editors, *ICCS-00*, volume 1867 of *Lecture Notes in Artificial Intelligence*, pages 290–303. SV, 2000.

[8] F. Baader and P. Narendran. Unification of Concept Terms in Description Logics. In *Proceedings of ECAI-98*, pp. 331–335, John Wiley & Sons Ltd., 1998.

[9] A. Borgida and D. L. McGuinness. Asking Queries about Frames. In *Proceedings of KR'96*, pp. 340–349, Morgan Kaufmann Publishers, 1996.

[10] William W. Cohen, Alex Borgida, and Haym Hirsh. Computing least common subsumers in description logics. In William Swartout, editor, *Proc. of the 10th Nat. Conf. on Artificial Intelligence (AAAI-92)*, pages 754–760, San Jose, CA, 1992. AAAI Press/The MIT Press.

[11] R. Küsters and A. Borgida. What‘s in an attribute? Consequences for the least common subsumer. *JAIR*, 14:167–203, 2001.

[12] R. Küsters and R. Molitor. Approximating most specifc concepts in description logics with existential restrictions. In *Proc. of the 24th German Annual Conf. on Artificial Intelligence (KI'01)*, 2001. to appear.

[13] D.L. McGuinness. *Explaining Reasoning in Description Logics.* PhD thesis, Department of Computer Science, Rutgers University, October, 1996.

[14] Bernhard Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence Journal*, 43:235–249, 1990.

[15] U. Sattler. *Terminological knowledge representation systems in a process engineering application.* PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, 1998

[16] A.-Y. Turhan and R. Molitor. Using lazy unfolding for the computation of least common subsumers. In *DL-2001*, 2001.

Discourse and Application Modeling for Dialogue Systems

Kerstin Bücher, Yves Forkl, Günther Görz, Martin Klarner, and Bernd Ludwig
Computer Science Institute and FORWISS, Erlangen, Germany
eMail: `goerz@informatik.uni-erlangen.de`

Abstract

Spoken – and even written – language dialogue systems become of increasing importance for various information gathering and device controlling tasks. With an example taken form the EMBASSI joint project, after a brief discussion of the linguistic processing part, we describe how description logics are used in modeling the application domain as well as the linguistic domain. Linking lexical semantics with application specific concepts is a nontrivial problem, in particular in cases where there is no direct correspondence. So, the paper ends presenting our approach to solve this linking problem.

1 Generic Dialogue Management in EMBASSI

1.1 Application Background: The EMBASSI project

To a large extent, our research and development work in the field of dialogue systems is done within the German joint project EMBASSI[1] which aims to provide easy access for everybody to complex technical systems (A/V home theatre, car devices and public terminals), encouraging multi-modal user input, e.g. speech, gestures, and pointing. Besides a speech parser, our contribution consists of, first, a dialogue manager, second, a formal ontology and third, a generation component to communicate system utterances to the user.

In our approach to dialogue management, which features declarative modeling of the system's evolving *information state* rather than following a simple FSA-based procedural strategy, a fine-grained and well-structured ontological

[1]"Elektronische Multimediale Bedien- und Service-Assistenz", sponsored by the German Fed. Ministry of Research

hierarchy of semantic concepts is extremely important to enable logical inferences on these concepts. We formalize them using Description Logics (DL)[2] since they suit our needs quite well.

The novel architecture of the EMBASSI project brings up two problems to be mastered in the *domain model* (based on the formal ontology): First, the processing of user input is separated from the execution of system operations by introducing an interface between the dialogue manager and the applications which are controlled by so-called “assistant modules”. These operations have to be derived from the user’s utterances by constructing and representing their semantics and linking them to application concepts. To accomplish this, messages have to be exchanged between the dialogue manager and the assistants. We chose A-Box statements in DL (embedded in KQML statements) as a formalism for these messages. Second, there is a great variety of application components by different manufacturers. The EMBASSI ontology functions as a standardized system-wide terminology encompassing the whole world of functions and objects referred to by all applications and assistants controlling several audio/video devices in the home environment. Each of their functions and objects must be both uniquely named and given a precise semantic definition as a *concept.*

1.2 System Architecture and Knowledge Representation

EMBASSI is implemented as a distributed system with several autonomous modules communicating among each other in order to exchange information about the current state of the interfaced applications. Modules make their internal state partially available to the dialogue system as far as it is necessary in order to continue a dialogue. The current state of the application is represented by an A-Box containing assertions about instances which refer to objects of the application.

For example, `AvEvents`[3] may be retrieved from a data base and assertions about them are eventually added to the A-Box if they become relevant for the dialogue.

In addition, the A-Box also contains instances for all actions which have been executed up to now by some module of the (multi agent) system. All actions which are primitive with respect to the application domain model are executed procedurally.

If a module executes an action, a transition from one state to another is achieved as adding the effects of the action changes the content of the A-Box and the facts which can be inferred.

For this purpose, application modules exchange information with the dialogue system via KQML messages containing assertions about application ob-

[2]For an introduction to DL see e.g. [Don96]

[3]instances of the concept which describes all kinds of audio and video transmissions.

jects and actions. In particular, messages about actions to be executed and the computed effects are communicated between the modules. In order to update its knowledge about the application state, the dialogue system adds the content of incoming messages to the appropriate A-Box.

In complete analogy to the description of the current application state, the current state of the dialogue is represented as an A-Box whose assertions are about instances from the discourse and linguistic domain. In this case, objects are so called discourse referents representing instances of linguistic concepts as introduced in utterances. Again, instances can assert facts about actions as well as objects. Primitive actions in this domain are performatives like *I want, I ask, I state, I command.* They are executed by the dialogue system and modify the state a dialogue is currently in. From this point of view, the dialogue system behaves like an application module using a different A-Box to represent the current state of affairs.

The task of the system as a whole is to compute whether utterances from the user are satisfiable in the given state of the application. When passed to the dialogue system by the parser, an utterance only affects the A-Box for the discourse domain as its semantic representation is added there. So, what is the relationship between the two A-Boxes? The idea is that assertions in the discourse A-Box have to be satisfiable with respect to the application A-Box (i.e. if the state of the application allows the satisfiability directly or after a sequence of appropriate actions in the application domain). For example, the utterance *Record the thriller this evening!"* requires that the application A-Box contains an instance of `AvEvent` satisfying the imposed constraints and that the preconditions for `Record` are satisfiable.

In the case of a unique satisfaction of the utterance, an application assistant executes the corresponding action and reports the status of the execution back to the dialogue manager (DM). If the action has been executed **successfully**, the result is communicated. In the **case of an error**, the user should be informed about the possible cause. Otherwise, the result consists of a number of **different alternatives** which are presented to the user. The content of the system responses is represented in terms of A-Box statements.

2 Organisation of the Domain Model

The formal ontology structuring the domain model is organised in three domain areas, reflecting a fundamental partitioning of the dialogue manager's "world": First, the **Linguistic Domain**, comprising the speech parser's linguistic knowledge, especially from lexical semantics. Second, the **Discourse Domain**, which contains internal concepts (independent of a specific input mode, but supporting poly-modal processing) of the dialogue manager. They reflect its actions and the kinds of objects processed in the context of interacting with the user and the core system. This domain, although of crucial importance for dialogue management, will not be described here because of lacking space. Third and most important, the **Application Domain**: This is the area where knowledge about the functionality of all applications, assistants and devices is held. It equals the "domain model" in the classical sense, i.e. a semantic description of the functions and objects pertaining to the domain of application. In contrast to the preceding two areas, the terminology defined here is shared among the dialogue manager and the assistants and applications (the core system), for they need this language to exchange commands and requests.

2.1 Modeling the Application Domain

The modeling of the application domain demands a joint effort of the ontology creators on the one side and the application and assistant developers on the other side, for only the latter have precise knowledge of the funcionality they offer in their component. Any component's function and its parameter types must be integrated correctly in the ontology. Concepts for actions are distinguished from those for objects – in analogy to the distinction between objects and their methods in object-oriented programming languages.

The application situation is represented in terms of A-Box assertions; so, the scenario is an extension of the domain model.

In order for the dialogue manager to reason with errors, any error statement must confine to notions that have precise semantics and are part of the ontology. Thus, the application and assistant developers must give a semantic definition for any error type that their component may signal to the dialogue manager.

As an example, we consider a HAVi[4]-programmable VCR. Commands available in the HAVi class VCR may be uttered in natural language to use the VCR. HAVi ist defined in terms of a Java class hierarchy for object and function types which are translated automatically into a hierarchy of DL concepts. Mandatory derivation of all classes had to start from a set of predefined ontology base classes representing the *Generic Base Model* which comprises the top-level concepts for a broad range of application domains.

Currently we translate OO-subclass relations into DL-subconcept relations. Field members of classes are translated into forall constraints in DL, as shown below. The signature of OO-methods is represented by embedding each method as a subconcept of a concept of a generic base model to be chosen manually. Parameters of of methods are represented in the DL translation as forall constraints. As we use CICLOP as inference tool, we translate enumeration types and constants into primitive concepts. As a consequence, we do not need high

[4]HAVi stands for *Home Audio Video Interoperability* and represents a common, openly-licensable specification for networking digital home entertainment products.

expressivity for the DL language. We are currently investigating the limits of this approach. For feature-value formalisms it seems to be sufficient.

As an example for the derivation of a fragment of the DL domain model consider the following class which is part of the of the HAVi VCR class:

```
class VCRObj{
   int id;
   play();
   record();
   variableForward(forwardSpeedLevel speed);
}
```

with `enum forwardSpeedLevel = {fastForward5, ...}` .

From the viewpoint of DL, the definition of `VCRObj` introduces a subconcept of `object` (class) and three subconcepts of `action` (methods):

$$\begin{array}{lcl}
\texttt{VCRObj} & := & \texttt{object} \sqcap \forall\texttt{has-id.int} \\
\texttt{VCRObj-play} & := & \texttt{action} \\
\texttt{VCRObj-record} & := & \texttt{action} \\
\texttt{VCRObj-variableForward} & := & \texttt{action} \sqcap \forall\texttt{has-speed.forwardSpeedLevel}
\end{array}$$

For `VCRObj-variableForward`, this means that it is an `action`, and its only parameter has-speed has to be a `forwardSpeedLevel`. This is essentially the semantics of the method definition above, but `VCRObj-variableForward` is permitted only for objects of class `VCRObj`. We can account for this by refining our definition of in the following way:

$$\begin{array}{lcl}
\texttt{VCRObj-variableForward} & := & \texttt{action} \sqcap \forall\texttt{has-subject.VCRObj} \\
 & & \qquad\quad \sqcap \forall\texttt{has-speed.forwardSpeedLevel}
\end{array}$$

Using these concept definitions, we are now able to infer the specific operation from user utterances like "schnell vorspulen!", the meaning of which is represented in the A-Box by a domain-instantiated DRS. So, there is a `VCRObj` l_1 in our scenario asserting $\texttt{VCRObj}(l_1)$.

The invocation of the method l_1`.variableForward(fastForward5)` is asserted by

$$\texttt{action}(\alpha) \wedge \texttt{has-subject}(\alpha, l_1) \wedge \texttt{has-speed}(\alpha, \texttt{variableForward5})$$

from which $\texttt{VCRObj-variableForward}(\alpha)$ is derivable.

At this point, we have to refine the domain model again. Up to now, it differentiates between several concepts for objects (e.g. `VCRObj`, `forwardSpeedLevel`), but `action` is the only primitive concept for actions. Therefore, all verbs in the lexicon would have the same meaning, namely `action`. Consequently, whatever verbs were used, always e.g. `VCRObj-play` would be inferred. To cope with this difficulty, `action` has to be refined by introducing primitive subconcepts of it that are distinct from each other in order to separate different verb meanings. E.g. we introduce `play` $\subseteq$ `action`, `record` $\subseteq$ `action` and `wind` $\subseteq$ `action`. These (distinct) subconcepts of `action` would allow us create the following entries:

abspielen	`play`		
aufnehmen	`record`	*aufzeichnen*	`record`
spulen	`wind`		

These lexical entries suffice to distinguish which methods are meant by a certain verb, if we refine the definitions for the methods of the class `VCRObj`:

$$\begin{array}{llll}
\texttt{VCRObj-play} & := & \texttt{play} & \sqcap\ \forall\texttt{has-subject.VCRObj} \\
\texttt{VCRObj-record} & := & \texttt{record} & \sqcap\ \forall\texttt{has-subject.VCRObj} \\
\texttt{VCRObj-variableForward} & := & \texttt{wind} & \sqcap\ \forall\texttt{has-subject.VCRObj} \\
 & & & \sqcap\ \forall\texttt{has-direction.direction} \\
 & & & \sqcap\ \forall\texttt{has-speed.forwardSpeedLevel}
\end{array}$$

Generalizing the example above, we have outlined how formal DL domain model parts may be derived from underlying class definitions for objects and methods used in the implementation of the application assistant.

The division of the domain model in a generic upper part of primitive concepts not specific for the application and a lower part of application relevant concepts allows to reuse the generic part ([Ram97]). So, the knowledge representation task for the design of a specific dialogue system is considerably simplified.

Under quantitative aspects, our DL model for the A/V home theatre domain currently contains 88 primitive and 24 complex concepts, whereas the car entertainment domain has 12 primitive and 47 complex concepts. Both domains taken together will soon dispose of about 500 complex concepts, due to the integration of increased application functionality and much more detailed linguistic modeling.

2.2 Modeling the Linguistic Domain

Our ultimate aim in interpreting an user utterance is to construct one or more hypotheses of its meaning in terms of the concept definitions from the domain model. In order to achieve this, we have to devise a method for mapping natural language utterances onto terms describing method invocations.

We use a lexicon for the vocabulary V specific to the domain in which each word w refers to one or more *lexical concepts* which serve several purposes. First of all, synonymous words will receive the same lexical concept. Second, it is possible to account for polysemous words by having multiple entries, each with a different associated lexical concept. Third, the linguistic modeling of relations expressed in utterances can dispense of application-specific characteristics. Last but not least: In the domain model, these lexical concepts form a hierarchy of their own which is defined independently of the hierarchy structuring the domain model in the narrower sense, i.e. that of the application domain. This helps avoid those very common problems arising from the mingling of cognitive-lexical and factual relations.

In order to model the lexical semantics in a way that is maximally independent from any particular application, we employ a hierarchy of semantic types which is implemented as a DL terminology. Types are represented as disjoint atomic concepts and assigned to elements in the lexicon. For example, we have the following assignment of surface word forms (lower case) to atomic lexical concepts (upper case):

```
sehen SEHEN, anschauen ANSCHAUEN, gucken GUCKEN,
aufnehmen AUFNEHMEN, aufzeichnen AUFZEICHNEN
Krimi SPARTE
```

To capture the inter-chunk semantics of an utterance, i.e. the meaning of the combination of chunks, we introduce complex concepts obtained from interpreting case frames which describe the syntactic and semantic relations between chunks as concept definitions. The idea is as follows: Given an excerpt of the case frame for “aufnehmen” (with the thematic role `patiens` and the `acc`usative case marker)

```
aufnehmen patiens SENDER | SENDUNG | SPARTE | TITEL
          acc
```

we allow four different concepts that my fill the thematic role *patiens* of the lexical concept of `AUFNEHMEN` representing the semantic type of `aufnehmen`. This informal concept definition is captured precisely in terms of the following $\mathcal{ALC}$ expression:

$$C \doteq \mathtt{AUFNEHMEN} \wedge \forall \mathtt{patiens}.(\mathtt{Sender} \vee \mathtt{Sendung} \vee \mathtt{Sparte} \vee \mathtt{Titel})$$

Such a concept definition can be used to determine whether two chunks can be combined consistently with the available lexical semantics:

`[den Krimi] [aufnehmen]`

Analyzing the syntactic features (accusative) of the two chunks given, one finds that `[den Krimi]` may potentially serve as *patiens* for `[aufnehmen]`. This fact is validated by computing that the concept resulting from the combination of the two chunks is consistent with the above definition:

$$\mathtt{AUFNEHMEN} \wedge \forall \mathtt{patiens}.\mathtt{Sparte} \subseteq C$$

2.3 Linking Lexical Semantics with Domain Pragmatics

For linking lexical semantics with an application specific domain model we have to master the following tasks: *First*, relating application concepts with lexical concepts. For linking notions of the application with semantic types, we first have to cope with the usage of synonyms in natural language for a uniquely determined application concept. And *second*, relating roles with thematic roles, which means reasoning about individuals, i.e. in the A-Box.

To continue the example of `[den krimi] [aufnehmen]`, we have a look at the DRS constructed by the parser that makes use of the case frame shown above:

$$\left[\begin{array}{l} a\ k\ \text{krimi} \\ \hline \texttt{AUFNEHMEN}(a) \\ \texttt{patiens}(a, k) \\ \texttt{SPARTE}(k) \\ \texttt{wert}(k, \text{krimi}) \\ \texttt{SPARTENWERT}(\text{krimi}) \end{array}\right]$$

As discussed above, the task to be carried out now is to link lexical semantics and the application domain model in order to construct a representation for the application specific meaning of this DRS (A-Box).

We have already mentioned that via $\forall$`lexconcept.AUFNEHMEN` and $\forall$`lexconcept.SPARTE` we find the application specific meaning of both lexical concepts by inferring their descendants in a given T-Box. The issue still to be addressed is how the roles are mapped onto the corresponding ones in the application domain. If `SENDUNG` was the only `patiens` for `AUFNEHMEN`, one possibility would be to apply a rule like following one in the A-Box as (variables x, y, u, v are all-quantified):

$$\texttt{patiens}(x, y) \wedge \texttt{lexconcept}(u, x) \wedge \texttt{SENDUNG}(y) \wedge \texttt{lexconcept}(v, y) \;\rightarrow\; \texttt{has-avevent}(x, y)$$

But the example shows that this is not sufficient, as

$$\texttt{patiens}(x, y) \wedge \texttt{lexconcept}(u, x) \wedge \texttt{SPARTE}(y) \wedge \texttt{lexconcept}(v, y) \;\rightarrow\; \texttt{has-avevent}(x, y)$$

would wrongly imply y to be an `AvEvent` instead of a `Genre`. Therefore, before letting such a rule fire one has to determine two things:

- What are concepts in the range of `has-avevent`, if we assume `Record` to be in its domain? As we find `AvEvent`, but not `Genre`, the case frame is admissible only, if `Genre` is a part of the definition for `AvEvent`. To state this more precisely:

- Is it possible – by using the transitivity of roles – to construct a role that has `AvEvent` in its domain and `Genre` in its range? Given our definition of `AvEvent` we find that $R := \texttt{has-bibdata} \circ \texttt{has-genre}$ is the desired solution.

Now we are allowed to introduce new individuals of the concepts "on the path from `AvEvent` to `Genre`". In the example, there is only one such concept, namely `BibData`. As a consequence, we get the following DRS that corresponds a DRS in the application domain model:

$$\left[\begin{array}{l} r\ \ e\ \ b\ \text{krimi} \\ \hline \texttt{Record}(r) \\ \texttt{has-avevent}(r, e) \\ \texttt{AvEvent}(e) \\ \texttt{has-bibdata}(e, b) \\ \texttt{BibData}(b) \\ \texttt{has-genre}(b, g) \\ \texttt{Genre}(g) \\ \texttt{value}(g, \text{krimi}) \\ \texttt{GenreValue}(krimi) \end{array}\right]$$

2.4 Conclusion

This papers presents an DL based approach on domain modelling for natural language dialogue systems. It devises a automatic approach to the translation of feature-value formalisms into DL. Most domains for automatic information services can be modelled in a very simple DL language. Therefore, the approach serves for automatic and efficient configuration of dialogue systems. We showed that another important point is the separation between application domain and dialogue domain which is obtained at the cost of manually linking lexical semantics with an application domain model. In this way it is much easier to reuse linguistic resources like lexicons and grammars for the purpose of configuration. Additionally, we can increase the robustness and naturalness of dialogue systems in comparison with other approaches basing on keyword spotting and finite state automata.

References

[Abn91] S. Abney, *Parsing By Chunks.* In: R. Berwick, S. Abney, C. Tenny (Eds.), *Principle-based Parsing*. Kluwer, 1991

[KaR93] H. Kamp, U. Reyle, *From Discourse to Logic*. Dordrecht: Kluwer, 1993

[CIC99] LIIA-ENSAIS, *CICLOP version 1.b3 User Manual*. Strasbourg, 1999. `http://massenet.u-strasbg.fr/LIIA/ciclop/ciclop.htm`

[Don96] F.M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, *Reasoning in Description Logics*. In: G. Brewka (editor), *Foundations of Knowledge Representation*. CSLI-Publications, 1996, 191–236

[Abn95] S. Abney, *Chunks and Dependencies: Bringing Processing Evidence to Bear on Syntax*. In: *Computational Linguistics and the Foundations of Linguistic Theory*. CSLI-Publications, 1995

[Kus96] S. Kuschert, *Higher Order Dynamics: Relating Operational and Denotational Semantics for λ-DRT*. CLAUS-Report 84, Saarbrücken, 1996

[Lit99] D. Litman, M. Walker, M. Kearns, *Acquiring Knowledge of System Performance for Spoken Dialogue*. In: *Proceedings of the IJCAI 99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*. Stockholm, Sweden, 1999, 73–80

[Ram97] A. Ramsay, *Does It Make Any Sense? Update Semantics as Epistemic Reasoning*. In: *Proceedings of ACL 97*. Morgan Kaufman Publishers, 1997

[Sut98] S. Sutton et al., *Universal speech tools: The CSLU toolkit*. In: *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP'98)*. Volume 7, Sydney, Australia, 1998, 3221–24

[Tra94] D. Traum, *A Computational Theory of Grounding in Natural Language Conversation*. Ph.D. Thesis, Computer Science Dept., University of Rochester, 1994

An Optimized Tableau Structure for Explicit Representation of Disjunction

François de Bertrand de Beuvron, Martina Kullmann
and François Rousselot
Laboratoire d'Informatique et d'Intelligence Artificielle,
ENSAIS, FRANCE
beuvron@liia.u-strasbg.fr

Abstract

Recent DL reasoners (Ciclop [1], DLP [8],iFact [5], RACER [2]) have proven their practical efficiency for concept expression satisfiability checking (TBox Reasonning) within expressive DLs. Typically, such reasoners explore one branch of disjunction (alternatives introduced by OR construct) at a time, backtracking when necessary. We present in this paper a new tableau structure that efficiently keep track of proof states for more than one alternative. We claim that such structure called *Alt*-Tableau is well suited for reasoning within ABoxes and so-called Concrete Domains [7]. We also present a decision procedure for the satisfiability of $\mathcal{ALC}$-concept within *Alt*-Tableau .

1 Informal Presentation and Motivations

The idea presented in this paper is orthogonal to expressivity, and can be applied to any logic that allow for general disjunction. However, if allowing cyclic TBoxes and/or high expressivity as SHIQ [5] we cannot rely on the finite tree model property, and tableau algorithm for these logics have to include some kind of so-called blocking mechanism. Intuitively, blocking technics try to find if an already existing individual can be used instead of creating a new individual while processing *individual-introducing* construction such as $\exists R.C$. We think that we can take advantage of the structure, designed for disjunction representation, presented in this paper to deal also with this problem by designing a rule like *choose an existing individual from a set of candidate blockers, and try to associate it with the individual to be created. If none such association leads to satisfiability, create a new individual.* The choice within candidates can be

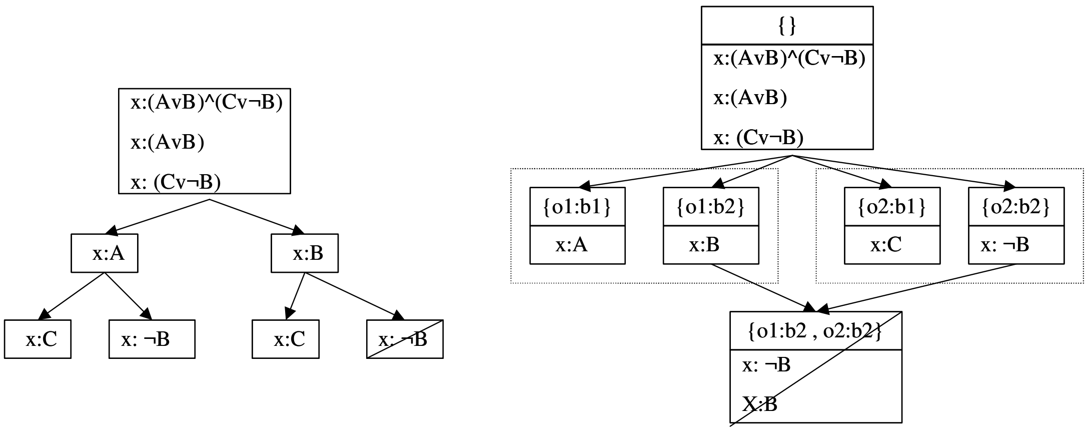


Figure 1: OR-tree and *Alt*-DAG for $\mathcal{ALC}$-concept $(A \sqcup B) \sqcap (C \sqcup \neg B)$

seen as a specific disjunction. Unfortunately, the formalization of this idea is still under study, and we will therefore restrict ourself to $\mathcal{ALC}$ with acyclic TBox (where satisfiablility can be reduced to $\mathcal{ALC}$-concept satisfiability by unfolding) in this paper.

Now consider the simple $\mathcal{ALC}$-concept $(A \sqcup B) \sqcap (C \sqcup \neg B)$. Three distinct tableaux can be obtained depending of the branches of ORs we choose. The fourth possibility lead to a clash. These tableaux can be represented by two OR-Trees (the first beeing given in figure 1 depending of the OR we choose to expand first. Note that this OR-Tree structure is effectively used in many reasoners although implicitely : only one branch of the OR-Tree is expanded at a given time, some backtracking being applied if the current branch lead to a clash. Nevertheless, it is easy to see that this is not optimal : in the first tree, two nodes correspond to the same expression C. Suppose that C is now some complex sub-expression, it will be expanded twice. Note that the $x : C$ constraint cannot be straightforwardly expanded once and used for both branches since this expansion may interact with the constraints $x : A$ and/or $x : B$. To avoid such redundancy in the structure proposed here, we create a single node (called BranchNode) for each branch of all disjunctions at a given level. If and only if there is some kind of interaction between branches introduced by distinct ORs, we create a new node (called MergeNode) representing the situation where both branches have been chosen. The resulting structure presented in figure 1, is no more a tree but an Acyclic Directed Graph we call *Alt*-DAG . This representation has two main advantages:

- each constraint is represented only once and will be expanded only once eventually producing new MergeNodes if it interact with other constraints in distinct alternatives.

- more than one alternative can be efficiently stored in the structure. Furthermore, it can be efficiently used in subsequent proofs allowing enhanced caching strategies. Also, this structure is well suited for designing parallel solvers.

We claim therefore that *Alt*-DAG representation is particularly well suited for the following tasks (see [6] for a detailed presentation):

- using heuristics to guide the proof : Some expansion rules are computationally more expansive than others. It may be effective to define heuristics to guide the proof. This is possible only if more than one alternative can be represented within the tableau, effectively allowing to switch from an alternative to another while keeping track of the proof state in each alternative. Although, please note that such improvement can only be obtained if more than one alternative is satisfiable. In particular, if the expression we are checking is inconsistent, all the alternatives have to be checked, and the computational overhead of the structure presented here may be an handicap in such situation. However, the advantages of the single expansion of constraints, and of potential for parallelization, still hold.
- ABox consistency checking :
  - many disjunction in distinct individuals are expected to be unrelated : suppose a library database with persons and books. As usual, persons can be men or women, and books may be real books or articles. Wether a person is man or woman do not interfere with it's hiring of books or articles. In this case, allowing for some kind of *local* disjunction representation, while ensuring that interactions will be correctly taken into account when needed should prove very effective.
  - without specific caching strategy, checking ABox consistency will become more and more computationally intensive as the number of individuals increase. DL systems incorporating ABoxes generally provide some caching strategy to deal with this problem. However, without explicit representation of disjunction, these caches are often limited to some deterministic subset of individual's descriptions.
  - inconsistency is uncommon and often correspond to user's misconception.
- reasoning with concrete domains : the use of heuristics and the single expansion of constraints may become more important if one want to deal with so-called concrete domains : various concrete domains (numeric constraints, time constraint, string constraint...) will have various complexity,

both theoretically and practically. Choosing a proof that minimize the calls to the most demanding concrete reasoners may greatly reduce the overall proof time.

2 $\mathcal{ALC}$ tableau algorithm

$\mathcal{ALC}$-concept are defined as usual from a set $\mathcal{CN}$ of concept names and a set $\mathcal{RN}$ of role names. We denote by $sub(C)$ the set of subconcept in C and by $\|C\|$ the size of an $\mathcal{ALC}$-concept C (see fig. 2).

syntax	semantics	subconcepts	size
$A \in \mathcal{CN}$	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	{A}	1
$\top$	$\Delta^{\mathcal{I}}$	$\{\top\}$	1
$\neg D$	$\Delta^{\mathcal{I}} - D^{\mathcal{I}}$	$\{\neg D\} \cup sub(D)$	$\|D\| + 1$
$D \sqcap E$	$D^{\mathcal{I}} \cap E^{\mathcal{I}}$	$\{D \sqcap E\} \cup sub(D) \cup sub(E)$	$\|D\| + \|E\|$
$D \sqcup E$	$D^{\mathcal{I}} \cup E^{\mathcal{I}}$	$\{D \sqcup E\} \cup sub(D) \cup sub(E)$	$\|D\| + \|E\|$
$\exists R.D$	$\{d \in \Delta^{\mathcal{I}} : R^{\mathcal{I}}(d) \cap D^{\mathcal{I}} \neq \emptyset\}$	$\{\exists R.D\} \cup sub(D)$	$\|D\| + 1$
$\forall R.D$	$\{d \in \Delta^{\mathcal{I}} : R^{\mathcal{I}}(d) \subseteq D^{\mathcal{I}}\}$	$\{\forall R.D\} \cup sub(D)$	$\|D\| + 1$

Figure 2: $\mathcal{ALC}$-concept

In the rest of this presentation, we will assume that $\mathcal{ALC}$-concept are finite and are in Negation Normal Form (NNF) in which negation apply only to concept names. The transformation of a $\mathcal{ALC}$-concept to a equivalent $\mathcal{ALC}$-concept in NNF is quite straightforward, and will be skipped in this paper.

2.1 *Alt*-Tableau

As foreseen in fig. 1, wee need to identify each expansion of disjunction. We will call OrIds these idendificators, and denote by $\mathcal{O}$ the set of OrIds. An alternative is a partial function from $\mathcal{O}$ to $\{0, 1\}$ identifying the branches of the disjunctions that have been expanded. We denote by $\mathcal{A} \subset 2^{\mathcal{O} \times \{0,1\}}$ the set of alternatives. An alternative a is said to be complete *iff* $domain(a) = \mathcal{O}$. Two alternatives a_1 and a_2 are compatibles noted $a_1 \odot a_2$ *iff* $\forall o \in domain(a_1) \cap domain(a_2), a_1(o) = a_2(o)$. Note that if two alternatives a_1 and a_2 are compatibles, then the relation $a_1 \cup a_2$ is also a partial function from $\mathcal{O}$ to {0,1} and so an alternative.

A pre-*Alt*-Tableau for an $\mathcal{ALC}$-concept C_0 is a quintuple $[S, \mathcal{O}, \mathcal{C}, \mathcal{LP}, \mathcal{E}]$ where S is the set of individuals, $\mathcal{O}$ is the set of OrIds, $\mathcal{C} \subseteq \mathcal{A}$ is the set of clashed alternatives, $\mathcal{LP} \subseteq \mathcal{A} \times S \times sub(C_0)$ keep track of the constraints associated with individuals, and $\mathcal{E} \subseteq \mathcal{A} \times S \times S \times \mathcal{RN}$ keep track of the relations between individuals. An *Alt*-Tableau is a sextuple $[S, \mathcal{O}, \mathcal{C}, \mathcal{LP}, \mathcal{E}, a_*]$, where $[S, \mathcal{O}, \mathcal{C}, \mathcal{LP}, \mathcal{E}]$ is a pre-*Alt*-Tableau and $a_* \in \mathcal{A}$ is a complete alternative. A pre-*Alt*-Tableau must verify properties A1-A7 of figure 3. An *Alt*-Tableau must also verify property A8.

Id	name	property definition
A1	C_0 in model	$\exists[\emptyset, s, C_0] \in \mathcal{LP}$
A$\neg$	$\neg$ coherence	if $[a_1, s, D] \in \mathcal{L}$ and $[a_2, s, \neg D] \in \mathcal{L}$ and $a_1 \odot a_2$ then $a_1 \cup a_2 \in \mathcal{C}$
A$\sqcap$	$\sqcap$ coherence	if $[a_1, s, D \sqcap E] \in \mathcal{L}$ and $a_1 \notin \mathcal{C}$ then $\exists a_2, a_3 \in \mathcal{A}$ with $a_2 \subseteq a_1$ and $a_3 \subseteq a_1$ such that $[a_2, s, D] \in \mathcal{L}$ and $[a_3, s, E] \in \mathcal{L}$
A$\sqcup$	$\sqcup$ coherence	if $[a_1, s, D \sqcup E] \in \mathcal{L}$ and $a_1 \notin \mathcal{C}$ then $\forall a_2 \in \mathcal{A}$ with a_2 complete and $a_1 \subseteq a_2$ $\exists a_3 \subseteq a_2$ such that $[a_3, s, D] \in \mathcal{L}$ or $[a_3, s, E] \in \mathcal{L}$
A$\forall$	$\forall$ coherence	if $[a_1, s_1, \forall R.D] \in \mathcal{L}$ and $[a_2, s_1, s_2, R] \in \mathcal{D}$ and $a_1 \odot a_2$ and $a_1 \notin \mathcal{C}$ and $a_2 \notin \mathcal{C}$ then $\exists a_3 \in \mathcal{A}$ with $a_3 \subseteq a_1 \cup a_2$ such that $[a_3, s_2, D] \in \mathcal{L}$
A$\exists$	$\exists$ coherence	if $[a_1, s_1, \exists R.D] \in \mathcal{L}$ and $a_1 \notin \mathcal{C}$ then $\exists a_2, a_3 \in \mathcal{A}$ with $a_2 \subseteq a_1$ and $a_3 \subseteq a_1$ such that $[a_2, s_2, D] \in \mathcal{L}$ and $[a_3, s_1, s_2, R] \in \mathcal{E}$
A7	clashes	$\forall a_1, a_2 \in \mathcal{A}, a_1 \in \mathcal{C} \wedge a_1 \subseteq a_2 \Rightarrow a_2 \in \mathcal{C}$
A8	satisfiable	$a_* \notin \mathcal{C}$

Figure 3: *Alt*-Tableau properties

Id	name	property definition
B1	C_0 in model	$\exists s \in S$ such that $C \in \mathcal{L}_b(s)$
B$\neg$	$\neg$ coherence	if $D \in \mathcal{L}_b(s)$ then $\neg D \notin \mathcal{L}_b(s)$
B$\sqcap$	$\sqcap$ coherence	if $D \sqcap E \in \mathcal{L}_b(s)$ then $D \in \mathcal{L}_b(s)$ and $E \in \mathcal{L}_b(s)$
B$\sqcup$	$\sqcap$ coherence	if $D \sqcup E \in \mathcal{L}_b(s)$ then $D \in \mathcal{L}_b(s)$ or $E \in \mathcal{L}_b(s)$
B$\forall$	$\forall$ coherence	if $\forall R.D \in \mathcal{L}_b(s_1)$ and $[s_1, s_2] \in \mathcal{E}_b$ then $D \in \mathcal{L}_b(s_2)$
B$\exists$	$\exists$ coherence	if $\exists R.D \in \mathcal{L}_b(s_1)$ then $\exists s_2 \in S$ such that $D \in \mathcal{L}_b(s_2)$ and $[s_1, s_2] \in \mathcal{E}_b$

Figure 4: *Basic*-Tableau properties

Lemma 2.1 *An $\mathcal{ALC}$-concept C_0 is satisfiable iff there exists an Alt-Tableau for C_0*

A tableau structure was introduced in [3] for ALC_{R^+}. Such a tableau called here *Basic*-Tableau to avoid confusion with *Alt*-Tableau is a triple $[S_b, \mathcal{L}_b, \mathcal{E}_b]$ where S_b is the set of individuals, $\mathcal{L}_b : S_b \rightarrow 2^{sub(C_0)}$ keep track of the constraints associated with individuals, and $\mathcal{E}_b : \mathcal{RN} \rightarrow 2^{S_b \times S_b}$ keep track of the relations between individuals. A *Basic*-Tableau must verify the properties given in figure 4. It is proven in [3] that an ALC_{R^+}-concept C_0 is satisfiable *iff* there is a *Basic*-Tableau for C_0. For proving lemma 2.1, we can prove that an *Alt*-Tableau exists for C_0 *iff* a *Basic*-Tableau exists for C_0.

Given an *Alt*-Tableau $\mathcal{T} = [S, \mathcal{O}, \mathcal{C}, \mathcal{L}, \mathcal{E}, a_*]$, we create a *Basic*-Tableau $\mathcal{T}_b = [S_b, \mathcal{L}_b, \mathcal{E}_b]$ with $S_b =$ S, $\mathcal{L}_b(s) = \{D \in sub(C_0)$ such that $\exists [a, s, C] \in \mathcal{L}$ for some $a \subseteq a_*\}$ and $\mathcal{E}_b(R) = \{[s_1, s_2] \in S \times S$ such that $\exists [a, s_1, s_2, R] \in \mathcal{E}$ for some $a \subseteq a_*\}$. Assuming that $\mathcal{T}$ verifies properties A1-A8 (fig 3) of *Alt*-Tableau , we prove below that $\mathcal{T}_b$ verifies properties B1-B$\exists$ (fig 4) of *Basic*-Tableau :

B1: obvious from A1

B$\neg$: $D \in \mathcal{L}_b(s) \Rightarrow \exists a_1 \subseteq a_*$ such that $[a_1, s, D] \in \mathcal{L}$
$\neg D \in \mathcal{L}_b(s) \Rightarrow \exists a_2 \subseteq a_*$ such that $[a_2, s, \neg D] \in \mathcal{L}$
since $a_1 \cup a_2 \subseteq a_*$ then (A$\neg$) $a_1 \cup a_2 \in \mathcal{C}$ so (A7) $a_* \in \mathcal{C}$
impossible by (A8). Hence $D \in \mathcal{L}_b(s) \Rightarrow \neg D \notin \mathcal{L}_b(s)$

B$\sqcap$: if $D \sqcap E \in \mathcal{L}_b(s)$ then ($\mathcal{L}_b$ def.) $\exists a_1 \subseteq a_*$ such that $[a_1, s, D \sqcap E] \in \mathcal{L}$ and (A7,A8) $a_1 \notin \mathcal{C}$
so (A$\sqcap$) $\exists a_2 \subseteq a_1 \subseteq a_*$ and $a_3 \subseteq a_1 \subseteq a_*$ such that $[a_2, s, D] \in \mathcal{L}$ and $[a_3, s, E] \in \mathcal{L}$. Hence ($\mathcal{L}_b$ def.) $D \in \mathcal{L}_b(s)$ and $E \in \mathcal{L}_b(s)$

B$\sqcup$: if $D \sqcup E \in \mathcal{L}_b(s)$ then ($\mathcal{L}_b$ def.) $\exists a_1 \subseteq a_*$ such that $[a_1, s, D \sqcup E] \in \mathcal{L}$ and (A7,A8) $a_1 \notin \mathcal{C}$
so (A$\sqcup$ + a_* complete) $\exists a_3 \subseteq a_*$ such that $[a_3, s, D] \in \mathcal{L}$ or $[a_3, s, E] \in \mathcal{L}$. Hence ($\mathcal{L}_b$ def.) $D \in \mathcal{L}_b(s)$ or $E \in \mathcal{L}_b(s)$

B$\forall$,B$\exists$: same kind of proof as B$\sqcap$

For the converse, we construct an *Alt*-Tableau from a *Basic*-Tableau with $S = S_b$, $\mathcal{O} = \emptyset$, $\mathcal{C} = \emptyset$, $\mathcal{L} = \{[\emptyset, s, D] \in \mathcal{A} \times S \times sub(C_0);\ D \in \mathcal{L}_b(s)\}$, $\mathcal{E} = \{[\emptyset, s_1, s_2, R] \in \mathcal{A} \times S \times S \times \mathcal{RN};\ [s_1, s_2] \in \mathcal{E}_b(R)\}$, and $a_* = \emptyset$. This tableau verifies properties A1-A8 (fig 3) :

A1: obvious from (B1) and $\emptyset \subseteq a_*$

A$\neg$: obvious from (B$\neg$) : the condition is never met

A$\sqcap$,A$\forall$,A$\exists$: obvious from (B$\sqcap$,B$\forall$,B$\exists$) : simply rewrite with $a_i = \emptyset$

A$\sqcup$: if $[a_1, s, D \sqcup E] \in \mathcal{L}$ then $a_1 = \emptyset$ and $D \sqcup E \in \mathcal{L}_b(s)$ so ($D \in \mathcal{L}_b(s)$ and $[\emptyset, s, D] \in \mathcal{L}$) or ($E \in \mathcal{L}_b(s)$ and $[\emptyset, s, E] \in \mathcal{L}$). since $\forall a, \emptyset \subseteq a$ A$\sqcup$ is trivially verified.

A8,A7: trivial since $\mathcal{C} = \emptyset$

2.2 *Alt*-Tableau Algorithm

The tableau algorithm presented below work on ProofContext . A ProofContext is a sextuple $[S, \mathcal{O}, \mathcal{C}, \mathcal{LP}, \mathcal{E}, \mathcal{P}]$ where S, $\mathcal{O}$, $\mathcal{C}$, $\mathcal{LP}$, and $\mathcal{E}$ are the same as for a pre-*Alt*-Tableau but without properties A1-A7, and $\mathcal{P} \subseteq \mathcal{A} \times S \times sub(C_0)$ will contains the set of already expanded Or constraint (see rule R$\sqcup$). To check the satisfiability of an $\mathcal{ALC}$-concept C_0, we apply the expansion rules of figure 5 to the initial ProofContext defined by : $S = \{s_0\}$, $\mathcal{O} = \emptyset$, $\mathcal{C} = \emptyset$, $\mathcal{L} = \{[\emptyset, s_0, C_0]\}$ (meaning individual s_0 must satisfy C_0 for the root alternative, and so for all alternatives), $\mathcal{E} = \emptyset$, $\mathcal{P} = \emptyset$.

A ProofContext is complete when none of the rules is applicable. Then, the algorithm return "C is satisfiable" if $\emptyset \in \mathcal{C}$, and "C is unsatisfiable" otherwise.

Lemma 2.2 (termination) *for every $\mathcal{ALC}$-concept C_0, the Alt-Tableau algorithm terminate*

This is ensured by the following properties of the *Alt*-Tableau algorithm :

T1 each constraint in $\mathcal{L}$ can trigger at most one rule expansion for rules R$\sqcap$, R$\sqcup$, and R$\exists$. Each pair of constraints in $\mathcal{L} \times \mathcal{E}$ can trigger at most one rule expansion for rule R$\forall$. This is ensured by conditions (2) on each rule.

T2 each rule expansion add at least one new element in $\mathcal{L}$, $\mathcal{E}$, or $\mathcal{C}$.

T3 S is given a tree structure by $\mathcal{E}$, since for all $s \in \mathcal{S}, s \neq s_0$ there is an unique $s_p \in S$ such that $[a, s_p, s, R] \in \mathcal{E}$ (see R$\exists$)

We denote by S^n the subset of S of depth n within the tree induced by $\mathcal{E}$. S^n is recursively defined by : $S^0 = \{s_0\}, S^{i+1} = \{s \in S; \exists[a, s_p, s, R] \in \mathcal{E}$ with $s_p \in S^i\}$. By analogy, we also define $\mathcal{L}^i = \{[a, s, C] \in \mathcal{L}; s \in S^i\}$, $\mathcal{E}^i = \{[a, s_1, s_2, R] \in \mathcal{E}; s_1 \in S^i\}$, $\mathcal{A}^i = \{a \in \mathcal{A}; \exists[a, s, C] \in \mathcal{L}^i\}$, $\mathcal{O}^i = \{o \in \mathcal{O}; \exists a \in \mathcal{A}^i$ such that $[o, 0] \in a$ or $[o, 1] \in a\}$.

T4 expansion of constraints $[a_1, s_1, C_1]$ by rules R$\sqcap$, R$\sqcup$, R$\forall$, and R$\exists$ can only produce new constraints $[a_2, s_2, C_2]$ with $\|C_1\| \leq \|C_2\| - 1$.

T5 Constraints in $\mathcal{L}^i$ can only be introduced by constraints in $\mathcal{L}^i$ by rules R$\sqcap$ and R$\sqcup$, constraints in $\mathcal{L}^{i-1}$ by rule R$\exists$, and constraints in $\mathcal{L}^{i-1}$ and $\mathcal{E}^{i-1}$ by rule R$\forall$. Constraints in $\mathcal{E}^i$ can only be introduced by constraints in $\mathcal{L}^{i-1}$ by rule R$\exists$.

T6 from (T4,T5), it is easy to see that

$$\max_{[a_1,s_1,C] \in \mathcal{L}^{i+1}} \|C\| \leq \max_{[a_2,s_2,C] \in \mathcal{L}^i} \|C\| \text{ and } \max_{[a_1,s_1,C] \in \mathcal{L}^0} \|C\| \leq \|C_0\|$$

Since only $\mathcal{ALC}$-concept of size at least two ($\exists R.C$) can produce new individuals, the depth of the tree is at most $\|C_0\| - 1$.

Id	name	rule definition
R$\neg$	clash creation	1- if $[a_1, s, D] \in \mathcal{L}$ and $[a_2, s, \neg D] \in \mathcal{L}$ and $a_1 \odot a_2$ 2- if $a_1 \cup a_2 \notin \mathcal{C}$ then $\mathcal{C} \rightarrowtail \mathcal{C} \cup \{a_1 \cup a_2\}$
R$\sqcap$	$\sqcap$-Rule	1- if $[a_1, s, D \sqcap E] \in \mathcal{L}$ and $a_1 \notin \mathcal{C}$ 2- if not $\exists a_2, a_3 \in \mathcal{A}$ with $a_2 \subseteq a_1$ and $a_3 \subseteq a_1$ such that $[a_2, s, D] \in \mathcal{L}$ and $[a_3, s, E] \in \mathcal{L}$ then $\mathcal{L} \rightarrowtail \mathcal{L} \cup \{[a_1, s, D], [a_1, s, E]\}$
R$\sqcup$	$\sqcup$-Rule	1- if $[a_1, s, D \sqcup E] \in \mathcal{L}$ and $a_1 \notin \mathcal{C}$ 2a- if not ($[a_1, s, D \sqcup E] \in \mathcal{P}$ or $\exists a_2 \in \mathcal{A}$ with $a_2 \subseteq a_1$ such that $[a_2, s, D] \in \mathcal{L}$ or $\exists a_2 \in \mathcal{A}$ with $a_2 \subseteq a_1$ such that $[a_3, s, E] \in \mathcal{L}$) then $\mathcal{P} \rightarrowtail \mathcal{P} \cup \{[a_1, s, D \sqcup E]\}$ create a new OrId o_n, $\mathcal{O} \rightarrowtail \mathcal{O} \cup \{o_n\}$ $\mathcal{L} \rightarrowtail \mathcal{L} \cup \{[a_1 \cup \{[o_n, 0]\}, s, D]\} \cup \{[a_1 \cup \{[o_n, 0]\}, s, E]\}$
R$\forall$	$\forall$-Rule	1- if $[a_1, s_1, u] \in \mathcal{LP}$ with $C/u = \forall R.D$ and $[a_2, s_1, s_2, R] \in \mathcal{E}$ and $a_1 \odot a_2$ 2- if not $\exists a_3 \in \mathcal{A}$ with $a_3 \subseteq a_1 \cup a_2$ such that $[a_3, s_2, D] \in \mathcal{L}$ then $\mathcal{LP} \rightarrowtail \mathcal{LP} \cup \{[a_1 \cup a_2, s_2, u \oplus [1]]\}$
R$\exists$	$\exists$-Rule	1- if $[a_1, s_1, u] \in \mathcal{LP}$ with $C/u = \exists R.D$ 2- if not $\exists a_2, a_3 \in \mathcal{A}$ with $a_2 \subseteq a_1$ and $a_3 \subseteq a_1$ such that $[a_2, s_1, s_2, R] \in \mathcal{E}$ and $[a_3, s_2, D] \in \mathcal{L}$ then create a new individual ns, $S \rightarrowtail S \cup \{ns\}$, $\mathcal{LP} \rightarrowtail \mathcal{LP} \cup \{[a_1, ns, u \oplus [1]]\}$, $\mathcal{E} \rightarrowtail \mathcal{E} \cup \{[a_1, s_1, ns, R]\}$
R6	clash propagation CP1	1- if $\exists a_1, a_2 \in \mathcal{C}$ such that $\exists a_3 \in \mathcal{A}, \exists o \in \mathcal{O}$ such that $a_1 = a_3 \cup \{[o, 0]\}$ and $a_2 = a_3 \cup \{[o, 1]\}$ 2- if $a_3 \notin \mathcal{C}$ then $\mathcal{C} \rightarrowtail \mathcal{C} \cup \{a_3\}$
R7	CP2	1- if $\exists a_1 \in \mathcal{C}$ such that $\exists a_2 \in \mathcal{A}$ such that $a_1 \subseteq a_2$ 2- if $a_2 \notin \mathcal{C}$ then $\mathcal{C} \rightarrowtail \mathcal{C} \cup \{a_2\}$

Figure 5: Expansion Rules for *Alt*-Tableau

T7 We denote by $nbrI(c)$ the maximum number of individuals that can be created in S^{i+1} by a constraint $c = [a, s, C] \in \mathcal{L}^i$. By recurrence on C we can prove that $nbrI([a, s, C]) \leq \|C\|$:

$$\begin{array}{rcl} A \in \mathcal{CN} & \rightarrow & nbrI(c) = 0 \\ D \sqcap E & \rightarrow & nbrI(c) = nbrI([a, s, D]) + nbrI([a, s, E]) \\ & & \text{(note that } \|D\| + \|E\| < \|C\|) \\ D \sqcup E & \rightarrow & nbrI(c) = nbrI([a, s, D]) + nbrI([a, s, E]) \\ \forall R.D & \rightarrow & nbrI(c) = 0 \text{ since R}\forall \text{ only create new constraints in } \mathcal{L}^{i+1} \\ \exists R.D & \rightarrow & nbrI(c) \leq 1 \text{ since constraints on D are in } \mathcal{L}^{i+1} \end{array}$$

Defining $nbrO(c)$ as the maximum number of OrIds that can be created by a constraint c, one can also prove by a similar argument that $nbrO([a, s, C]) \leq \|C\|$.

T8 From (T5), constraints created in $\mathcal{L}^{i+1}$ from $\mathcal{L}^i$ have the form $[a, s, D] \in \mathcal{A}^i \times S^{i+1} \times sub(C_0)$. There is at most $|\mathcal{A}^i|.|S^{i+1}|.\|C_0\|$ such constraints. From (T7) the expansion of each such constraints can create at most $\|C_0\|$ new individuals in S^{i+1} and also at most $\|C_0\|$ new OrIds in $\mathcal{O}^{i+1}$. Hence, $|S^{i+2}| \leq |\mathcal{A}^i|.|S^{i+1}|.\|C_0\|^2$ and $|\mathcal{O}^{i+1}| \leq |\mathcal{O}^i| + |\mathcal{A}^i|.|S^{i+1}|.\|C_0\|^2$. From the initialization, and (T7) it is easy to see that $|S^0| = 1$, $|\mathcal{O}^0| \leq \|C_0\|$, and $|S^1| \leq \|C_0\|$. Since the depth of the tree is finite (by T6), the sets $\mathcal{O}$ and S are finite, hence the sets $\mathcal{A}$, $\mathcal{C}$, $\mathcal{L}$, $\mathcal{E}$ and $\mathcal{P}$.

The maximum number of elements in a ProofContext is finite (T8). Each rule application adds at least one element in the ProofContext (T2). Hence the algorithm terminate.

Lemma 2.3 *when the algorithm terminates, the ProofContext is a pre-Alt-Tableau*

This is quite obvious since properties A$\neg$,A$\sqcap$,A$\forall$,A$\exists$ can be proven from the corresponding expansion rules. Property A1 is ensured by the initialisation of the ProofContext . Finally, property A7 is straightforwardly deduced from R$\neg$. Property A$\sqcup$ is deduced from rule R$\sqcap$ as follow :

A$\sqcup$: if $[a_1, s, D \sqcup E] \in \mathcal{L}$ and $a_1 \notin \mathcal{C}$ then either

- $[a_1, s, D \sqcup E] \in \mathcal{P}$, so the rule has been expanded, and $\exists o \in \mathcal{O}$ such that $[a_1 \cup \{[o, 0]\}, s, D] \in \mathcal{L}$ and $[a_1 \cup \{[o, 1]\}, s, D] \in \mathcal{L}$. Then for a complete a_2 such that $a_1 \subseteq a_2$, either $a_1 \cup \{[o, 0]\} \subseteq a_2$ or $a_1 \cup \{[o, 1]\} \subseteq a_2$
- $[a_1, s, D \sqcup E] \notin \mathcal{P}$, then from rule R$\sqcup$ ($\exists a_2 \in \mathcal{A}$ with $a_2 \subseteq a_1$ and $[a_2, s, D] \in \mathcal{L}$) or ($\exists a_2 \in \mathcal{A}$ with $a_2 \subseteq a_1$ and $[a_2, s, E] \in \mathcal{L}$). For all complete $a_3 \in \mathcal{A}$, $a_1 \subseteq a3 \Rightarrow a_2 \subseteq a3$.

Lemma 2.4 *In a complete ProofContext , if $\emptyset \notin \mathcal{C}$ then there is a complete alternative a_* such that $a_* \notin \mathcal{C}$*

from rule R6, it is easy to prove that if $\forall a_1 \in \mathcal{A}$ with $card(a_1) = n$, $a_1 \in \mathcal{C}$ then $\forall a_2 \in \mathcal{A}$ with $card(a_2) = n-1$, $a_2 \in \mathcal{C}$: given a with $card(a) = n-1 < card(\mathcal{O})$, suppose $o \in \mathcal{O}$ such that $[o,0] \notin a$ and $[o,1] \notin a$ then
$\exists a_1, a_2 \in \mathcal{A}$ with $card(a_1) = card(a_2) = n$ such that $a_1 = a \cup \{[o,0]\}$ and $a_1 = a \cup \{[o,1]\}$. Since $a_1, a_2 \in \mathcal{C}$ by hypothesis, $a \in \mathcal{C}$ by rule R6. So (by recurrence on n), if all complete alternatives are in $\mathcal{C}$, the empty alternative is also in $\mathcal{C}$.

Lemma 2.5 (Soundness) *When the algorithm terminate, if $\emptyset \notin \mathcal{C}$ then the ProofContext is an Alt-Tableau .*

Direct consequence of lemma 2.3 and 2.4

Lemma 2.6 (Completeness) *If the $\mathcal{ALC}$-concept C_0 has an Basic-Tableau $\mathcal{T}_b$, the ProofContext $\mathcal{P}$ constructed by the algorithm is an Alt-Tableau .*

By lemma 2.3 $\mathcal{P}$ is in fact a pre-*Alt*-Tableau . In order to find a not clashing complete alternative a_* in $\mathcal{P}$, we can map the individuals in $\mathcal{P}$ to individuals in $\mathcal{T}$ in such a way (following the method given in [4]) that the constraints on individuals in $\mathcal{P}$ are a subset of the constraints on the mapped individuals in $\mathcal{T}$. We construct an alternative a_+ and a function $\pi : S \to S_b$ such that :

$$(*) \left\{ \begin{array}{l} \forall [a_1, s, C] \in \mathcal{L} \text{ with } a_1 \odot a_+ : D \in \mathcal{L}_b(\pi(s)) \\ \forall [a_1, s_1, s_2, R] \in \mathcal{E} \text{ with } a_1 \odot a_+ : [\pi(s_1), \pi(s_2)] \in \mathcal{E}_b \end{array} \right|$$

The (*) property can be preserved when applying expansion rules :

R$\forall$ if $[a_1, s, D \sqcap E] \in \mathcal{L}$ and $a_1 \odot a_+$, then $D \sqcap E \in \mathcal{L}_b(\pi(s))$ then (B$\sqcap$) $D \in \mathcal{L}_b(\pi(s))$ and $E \in \mathcal{L}_b(\pi(s))$. Applying rule R$\sqcap$ produce $[a_1, s, D]$ and $[a_2, s, E]$, so property (*) is not violated.

R$\sqcup$ if $[a_1, s, D \sqcup E] \in \mathcal{L}$ and $a_1 \odot a_+$, then $D \sqcup E \in \mathcal{L}_b(\pi(s))$ then (B$\sqcup$) $D \in \mathcal{L}_b(\pi(s))(1)$ or $E \in \mathcal{L}_b(\pi(s))(2)$. Suppose R$\sqcap$ is applied, producing $[a_1 \cup \{[o_n, 0]\}, s, D]$ and $[a_1 \cup \{[o_n, 1]\}, s, E]$.
Suppose (1) hold, then by setting $a_+ \leftarrow a_+ \cup \{[o_n, 0]\}$ property (*) is preserved since $a_1 \cup \{[o_n, 1]\}$ is not compatible with a_+, hence only D is required in $\mathcal{L}_b(\pi(s))$. Note also that all constraints with alternative a compatible with the old value of a_+ are still compatible with the new value since o_n is a fresh OrId. Also, all constraints with alternative a not compatible with the old value of a_+ are obviously not compatible with the new value. So modifying a_+ in the way above do not affect property (*) for previously expanded constraints.
Suppose (2) old, setting $a_+ \leftarrow a_+ \cup \{[o_n, 1]\}$ preserves property (*) by a similar argument.

R$\exists$ if $[a_1, s, \exists R.D] \in \mathcal{L}$ and $a_1 \odot a_+$, then $\exists R.D \in \mathcal{L}_b(\pi(s))$ then (B$\exists$) $\exists y \in S_b$ such that $[\pi(s), y] \in \mathcal{E}_b$ and $D \in \mathcal{L}_b(y)$.
Applying R$\exists$ generates a new variable $t \in S$ and the constraints $[a_1, t, D] \in \mathcal{L}$ and $[a_1, s, t, R] \in \mathcal{E}$. Setting $\pi \leftarrow \pi \cup [t, y]$ yields a function satisfying property (*)

R$\forall$ if $[a_1, s, \forall R.D] \in \mathcal{L}$ and $a_1 \odot a_+$, then $\forall R.D \in \mathcal{L}_b(\pi(s))$. If $[a_2, s, t, R] \in \mathcal{E}$ and $a_2 \odot a_+$, then $\pi(s), \pi(t)] \in \mathcal{E}_b$. Hence, by rule (B$\forall$), $D \in \mathcal{L}_b(\pi(t))$. Hence applying R$\forall$, producing $[a_1 \cup a_2, t, D]$, do not violate property (*).

Initialy, we set $a_+ = \emptyset$, and $\pi = \{[s_0, x_0]\}$ for some $x_0 \in S_b$ with $C_0 \in \mathcal{L}_b(x_0)$ (x_0 exists since $\mathcal{T}_b$ is a *Basic*-Tableau). Property (*) is verified.

When the algorithm terminate, suppose $a_+ \in \mathcal{C}$ then, since an alternative can only produce clashes (by rule R$\neg$, R6, R7) in compatible alternatives :

$$\begin{aligned} a_+ \in \mathcal{C} \quad &\Rightarrow \quad \exists a_1, a_2 \text{ such that } a_1 \odot a_2 \text{ and } a_1 \odot a_+ \text{ and } a_2 \odot a_+ \text{ and} \\ & \quad [a_1, s, A] \in \mathcal{L} \text{ and } [a_2, s, \neg A] \in \mathcal{L} \\ &\Rightarrow \quad (\text{since } a_1 \odot a_+ \text{ and } a_2 \odot a_+) A \in \mathcal{L}_b(\pi(s)) \text{ and } \neg A \in \mathcal{L}_b(\pi(s)) \end{aligned}$$

This is impossible since $\mathcal{T}_b$ is a *Basic*-Tableau . So $a_+ \notin \mathcal{C}$. By R6,R7 (see lemma 2.4) $\exists a_*$ such that a_* is complete and $a_+ \subseteq a_*$ and $a_* \notin \mathcal{C}$

Theorem 2.1 *The algorithm on Alt-Tableau presented in this paper is a decision procedure for the satisfiability of $\mathcal{ALC}$-concept*

This theorem is a direct consequence of lemma 2.1, 2.2, 2.5, and 2.6.

3 Conclusion

We have presented a new tableau structure and proved its applicability to DL by providing a sound and complete algorithm for $\mathcal{ALC}$-concept satisfiability checking relying on such structure. Obviously, many extensions are still to be defined to demonstrate the claims that it will be well suited for proof optimization, in particular with ABoxes and Concrete Domains. Nevertheless, this *Alt*-Tableau algorithm has been implemented in the CICLOP reasoner, and preliminary tests with toy examples are promising. An in-depth study with both constructed and real TBoxes and ABoxes have still to be carried out.

References

[1] F. de Beuvron, M. Kullmann, D. Rudloff, M. Schlick, and F. Rousselot. The description logic reasoner ciclop (version 2.0). In *Proceedings of the 7th Workshop on Automated Reasoning*, London, Great Britain, 2000.

[2] Volker Haarslev and Ralf Mller. Description of the racer system and its applications. In *Proceedubgs International Workshop on Description Logics (DL-2001)*, Stanford, USA, 2001.

[3] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics.* PhD thesis, University of Manchester, 1997.

[4] I. Horrocks and U. Sattler. A description logic with transitive and converse roles and role hierarchies. In *Proceedings of the International Workshop on Description Logics*, Povo - Trento, Italy, 1998. IRST.

[5] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, September 1999.

[6] Martina Kullmann. *Description Logic Models for Supporting Decision Making.* PhD thesis, Université Louis Pasteur, France, 2001.

[7] C. Lutz. Nexptime-complete description logics with concrete domains. In *Proceedings of the International Joint Conference on Automated Reasoning*, LNAI, Siena, Italy, 2001. Springer Verlag.

[8] P. F. Patel-Schneider. Dlp system description. In E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors, *Collected Papers from the International Description Logics Workshop (DL'98)*, pages 87–89, 1998.

A Knowledge Based System for Person-to-Person E-Commerce

Eugenio Di Sciascio, Francesco M.Donini, Marina Mongiello, Giacomo Piscitelli
Dipartimento di Elettrotecnica ed Elettronica
Politecnico di Bari
Via Re David, 200 – 70125 BARI, Italy
e-mail: {disciascio,donini,mongiello,piscitel}@poliba.it

Abstract

Short-term peer-to-peer (or person-to-person) e-commerce calls for an infrastructure treating in a uniform way supply and demand, which should base the matchings on a common ontology for describing both supply and demand.

Knowledge representation — in particular description logics — can deal with this uniform treatment of knowledge from vendors and customers, by modelling both as generic concepts to be matched.

We propose a logical approach to supply-demand matching in person-to-person e-commerce, which is deployed in a prototype system implemented for a particular case study but easily generalizable and is based on Classic, a well known knowledge representation system. Limits and advantages are analyzed and discussed.

1 Introduction

Electronic commerce differs from traditional commerce primarily in the way that information is exchanged and processed. Personal contacts, or at most contacts mediated through the phone or postal system, are or should be, substituted by information exchanged between computers flowing along digital networks. E-commerce can anyway mean several things.

In this paper we focus on what is known as Consumer-to-consumer e-commerce. This category, which we are more likely to call person-oriented or Person-to-Person (P2P) in accordance with the proposal in [4], is emerging slowly, as it is not backed up by the interests of large companies, but we believe it has great potentials for widespread diffusion. We basically consider the P2P scenario an electronic marketplace, or better an agorá[1], where peer consumers may propose their goods and products and dynamically deal with counteroffers or further specifications.

Our research tries to fit an existing gap in present deployment of P2P e-commerce, by extending the technology of search and of descriptions of searched items, in order to ease and improve the free play of demand and supply.

Most of current consumer or person oriented e-commerce strives to reproduce, with limited success, the usual commercial interaction taking place in physical stores. On one hand, consumers are reached by blind advertising on store products, and brand names — which are built out of massive economic investments — fill customers' lack of information. On the other hand, (virtual) stores are visited by those consumers who are interested in buying a product they think is available at that store. Portals reproduce specialized catalogues and consumer magazines.

We believe that this situation is not the necessary picture of e-commerce; it depends on the available technology for publishing and searching on the web.

Publishing on the Web is in fact a form of blind advertising of products. Although not as intrusive as spamming, the publisher has no precise idea of who is going to access the published information. Producers rely on keywords, and on indexing on prominent search engines, to be found by potential customers. Keywords are affected by the wide-experienced problems of both synonyms, and polysemic words. Categorization by prominent index sites may help some producers to be effectively found by potential customers; however, on the customer side, relying on indexes may result in a loss of useful information about small-size producers, or short-time suppliers. This is particularly relevant for agricultural products: supplies tend to have a seasonal time window, and different producers may alternate on the market, each one without the sufficient economic strength to put up a long-term, visible and reliable site.

On the customer side, there is no way to reach a product but by active search using the available technology, which penalizes P2P in favor of B2C. The use of main portals and indexes to speed up search tends to favor long-term renowned brands, in comparison with short-term supplies by small producers. This may result in a loss of competitive offers, which could have been economically more convenient just because of either geographical proximity, or smaller advertising overhead on the overall price.

One may be tempted to solve the problem of P2P by resorting to presently available Database technology. A site specialized in particular products

[1]Agorá: An assembly; hence, the place of assembly, especially the market place, in an ancient Greek city.

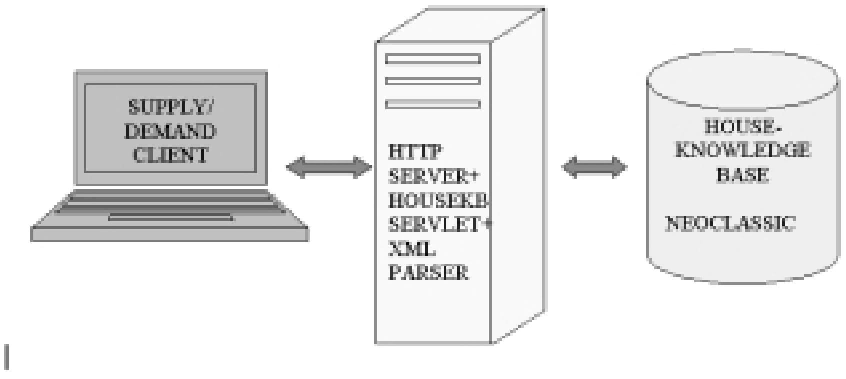


Figure 1: Three-tier architecture of the prototype system.

may help suppliers to dynamically insert and delete offers for definite time slots. The site could help also customers to issue queries regarding particular products, probably with the help of some interface to formulate queries that can be actually answered — using the right vocabulary, the right level of abstraction. To clarify with an example, an interface may help a user wondering, "Can I ask about 'merlot grapes', or should I use 'red grapes', or just 'grapes', and scrutinize retrieved answers after?". In a nutshell, the interface for queries should clarify to the user the Database Ontology (names and relations in the Database schema).

However, such a database technology solution captures only half of the matching of supply and demand in P2P e-commerce. In fact, it captures the request of a customer as a query, to be answered in a web-based database of offers by suppliers. But on the other hand, also the offer of a supplier can be treated as a query for interested customers in a database of customer requests, or even personal profiles. The choice of which is the data, and which is the query, depends here just on a point of view, and maybe, on who is more interested in actively finding the answer — and that is not necessarily the customer.

Summarizing, we believe that in small-size, short-term P2P commerce there is the need of an infrastructure treating in a uniform way suppliers and demanders, which should base the matchings on a common ontology for describing both supplies and demands.

Knowledge representation — in particular description logics — can deal with this uniform treatment of knowledge from suppliers and customers, by modelling both as generic concepts to be matched. The matching could be accomplished by a classification on the hierarchy of concepts, that forms the common ontology. Both supplier offers and customer requests could be classified, leading to exact match when they describe the same concept, and to partial match (still meaningful) when one is a subconcept of the other or concepts are compatible. For instance, an offer of 'red grapes' partially matches a request for 'merlot grapes'. The partial match can then be further investigated, still with the help of the infrastructure.

In fact, the logical approach — which Description Logics are based upon — allows for an open-world assumption. Incomplete information is allowed (and can be filled after a selection of possible matches), and absence of information can be distinguished from negative information, allowing to discard offers/requests without the necessary properties, and to ask for missing information in the potential matches.

Here we propose an approach to P2P e-commerce based on Description Logics and describe its implementation in a prototype system for a virtual brokering agency for apartments rental. Limits and possibilities are also addressed.

2 A logical approach to matching in P2P e-commerce

We now apply the framework of Description Logics to supply-demand matching in P2P e-commerce marketplaces. We suppose for simplicity a basic P2P e-commerce setting, in which every transaction involves just one supplier and one customer.

We formalize both supplies and demands as concept names (of two distinct alphabets), which share concepts and roles of a common product ontology. Hence, we assume the existence of two disjoint alphabets of supply names $\mathcal{C} = \{C_1, \ldots, C_n\}$, and demand names $\mathcal{D} = \{D_1, \ldots, D_m\}$. Moreover, we assume that the domain of interest is modeled with concepts and roles as in a generic Description Logic.

E.g., proposing the supply of an apartment in Brooklyn Heights, NY, could be described by

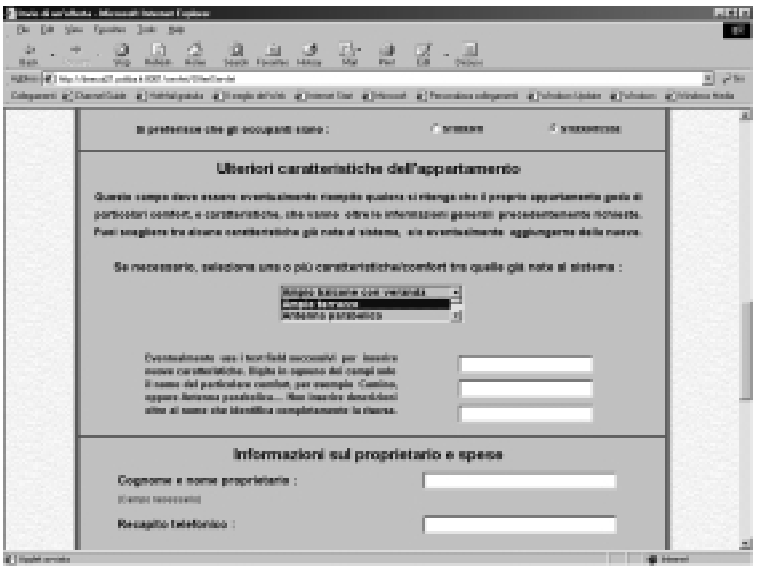
$$C_1 \doteq \texttt{apartment} \sqcap \exists \texttt{location.Brooklyn-Heights} \sqcap (\geq\ 2\ \texttt{rooms})$$

while two demands in the same domain could be described by

$$\begin{aligned} D_1 &\doteq \texttt{apartment} \sqcap \exists \texttt{options.Back-Yard-Garden} \\ D_2 &\doteq \texttt{apartment} \sqcap (\leq\ 1\ \texttt{rooms}) \end{aligned}$$

The interpretation of supply and demand concepts is on a single, common domain of possible transactions; the set of transactions compatible with them. When a transaction involves a single traded good — such as apartments — supplies and demands can be interpreted directly as sets of goods that can be exchanged. However, we prefer to interpret concepts as transactions since there are exchanges involving continuous goods, such as fruits, oil, etc.

Definition 1 *An offer C and a request D are [in]compatible iff the concept $C \sqcap D$ is [un]satisfiable.*



Si preferisce che gli occupanti siano : [illegible] [illegible]

Ulteriori caratteristiche dell'appartamento

Questo campo deve essere eventualmente riempito qualora si ritenga che il proprio appartamento goda di particolari comfort, o caratteristiche, che vanno oltre le informazioni generali precedentemente richieste. Puoi scegliere tra alcune caratteristiche già note al sistema, e/o eventualmente aggiungerne delle nuove.

Se necessario, seleziona una o più caratteristiche/comfort tra quelle già note al sistema :

Ampio balcone con veranda
[illegible]
Antenna parabolica

Eventualmente usa i text field successivi per inserire nuove caratteristiche. Digita in ognuno dei campi solo il nome del particolare comfort, per esempio Camino, oppure Antenna parabolica... Non inserire descrizioni oltre al nome che identifica completamente la risorsa.

Informazioni sul proprietario e spese

Cognome e nome proprietario :
(Campo necessario)

Recapito telefonico :

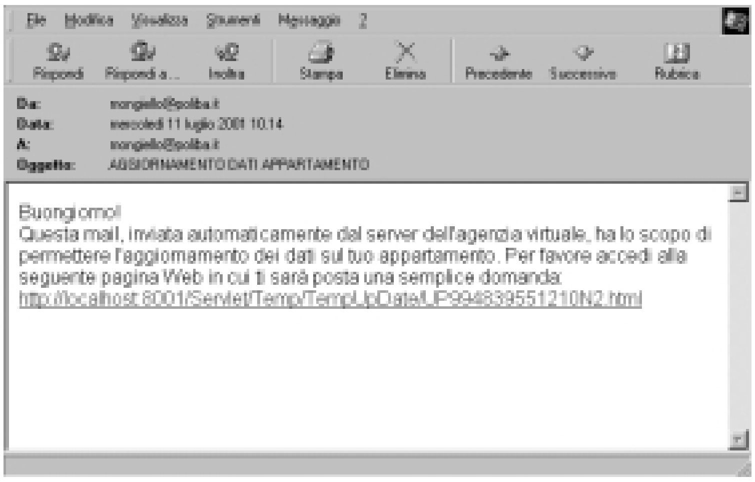
Figure 2: The form specifies the basic features usually requested to describe an apartment, such as price, number of rooms and so on. Further features/concepts, originally not foreseen and now part of the system knowledge, can be selected from the scrollable window. New concepts can be inserted in the textfields to extend the description of the apartment. After the insertion, the knowledge base is automatically updated. The new concepts appear in the list of features in the scrollable window and are available for further searches or insertions.

For the above examples, D_1 is compatible with C_1, and further inquiries could be issued to both parts to see if the transaction can be concluded. Instead, C_1 is incompatible with D_2, so this matching can be discarded in a first filtering phase.

There are cases, however, in which we can establish from known data that the transaction is already possible: *e.g.*, a demand of exotic fruits is fulfilled by a supply of bananas. Obviously, to establish this conclusion the system must have knowledge (through an inclusion assertion) that bananas are exotic fruits.

Definition 2 *A demand D is* fulfilled *by a supply C in a TBox T if C is subsumed by D in T.*

Unmatched demands and supplies can be “stored” as new concepts, and classified accordingly in the TBox. As soon as a matching supply/demand enters the system, the old request can be reexamined. In this sense, the approach based on Description Logics is balanced between supply and demand, and much different from usual B2C portals where a customer



Da: mongiello@poliba.it
Data: mercoledì 11 luglio 2001 10.14
A: mongiello@poliba.it
Oggetto: AGGIORNAMENTO DATI APPARTAMENTO

Buongiorno!
Questa mail, inviata automaticamente dal server dell'agenzia virtuale, ha lo scopo di permettere l'aggiornamento dei dati sul tuo appartamento. Per favore accedi alla seguente pagina Web in cui ti sarà posta una semplice domanda:
http://localhost:8001/Servlet/Temp/TempUpDate/UP994839551210N2.html

Figure 3: The text of an e-mail automatically sent to the owner of an apartment. The message points to a URL where the owner is asked to answer about the presence of features not included in the original description.

has to make active efforts to find the goods s/he looks for, and if she does not find them now, s/he has to re-enter data next week.

3 A case study: renting apartments to students

3.1 Scenario

Differently from most other countries, Italian universities do not have dormitories for their students, which are only provided to a minimum percentage of them. Also, often faculties are spread in different parts of towns and not concentrated in campuses.

This situation leads to a flourishing activity of privately owned apartments rental. Furthermore largest part of students do not rent an entire apartment or house, instead they are more likely to ask for a room and often for just a bed within a room.

Supply and demand are extremely dynamic and currently there are no brokering agencies, at least in most italian towns. Therefore, although there is a noteworthy aggregate turnover, typically supply meets demand in naive ways, *e.g.*, advertisements pinpointed on show-cases, word passed round, etc.

This business scenario can hence be considered an ideal candidate for a web-based system designed for person-to-person small business e-commerce whose objective is becoming a virtual brokering agency endowed of innovative services able to ease use and interaction, and closer to user needs than most of business-to-consumer e-malls are.

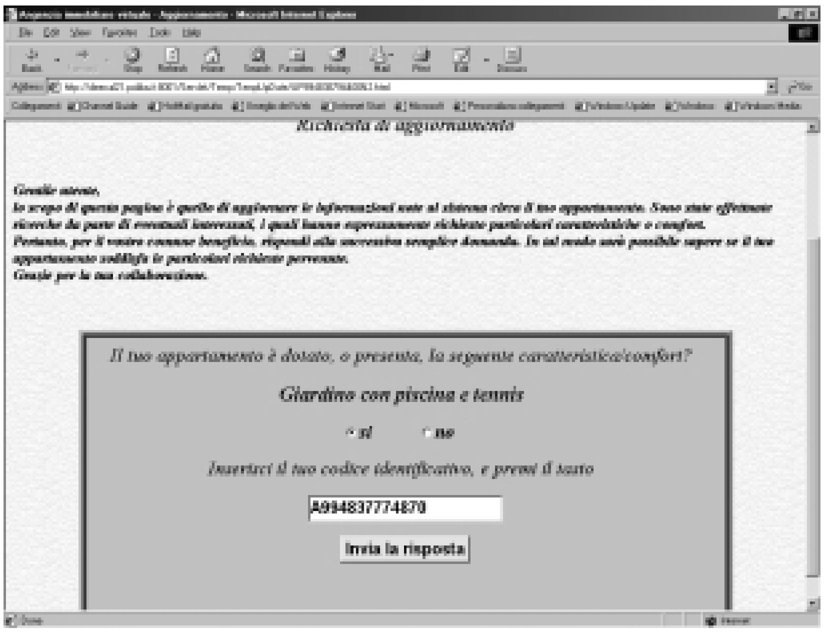
Main services we considered objective of our case study are:

1. Support to the user in the data insertion and query submission. The user is incrementally guided in the definition of a query or an offer. We remark here that though we distinguish for simplicity the supplier and the demander they are treated in a uniform way by the system.
2. Automatic construction and verification of satisfiability of the query.
3. Deduction of new knowledge on the basis of available data. As an example, the system automatically detects apartments available in a convenient area once the faculty has been entered.
4. Ability to provide conceptually approximate answers in the presence of unsatisfiable queries. Notice that this the way a human clerk behaves: when a request can not be satisfied he/she will propose the closest alternatives to the client, will not answer "no match".
5. Ability to manage incomplete queries and possibility to ask for unforeseen (hence not immediately available) services and features to the supplier.
6. Storage of satisfiable queries/demands that were still unmatched, with automatic reexamination when new supplies are provided, and notification on succesful match between supply and demand. The same service is available for unmatched supplies.

The remaining of this section describes main features of the implemented system. It is anyway noteworthy that, though the system has been actually designed as a virtual brokering acengy for apartments rental, similar issues exist in several other business scenarios. A nice property of our approach is in fact that reverting to other scenarios is quite simple, provided that basic concepts and a simple ontology are available. The system will extend its knowledge as more information becomes available without having to modify the structure of the system.

3.2 The virtual brokering agency

The proposed prototype system is based on a NeoClassic engine interfaced to the web via Java servlet technology. Classic is a member of the KL-ONE family of knowledge representation languages [2]. NeoClassic, a Classic implementation, is basically a knowledge server to build, manage, infer about, and query a classification hierarchy. NeoClassic is a frame-based knowledge representation system that differentiates between



Richiesta di aggiornamento

Gentile utente,
lo scopo di questa pagina è quello di aggiornare le informazioni note al sistema circa il tuo appartamento. Sono state effettuate ricerche da parte di eventuali interessati, i quali hanno espressamente richiesto particolari caratteristiche o confort.
Pertanto, per il vostro comune beneficio, rispondi alla successiva semplice domanda. In tal modo sarà possibile sapere se il tuo appartamento soddisfa le particolari richieste pervenute.
Grazie per la tua collaborazione.

Il tuo appartamento è dotato, o presenta, la seguente caratteristica/confort?

Giardino con piscina e tennis

[✓ sì] no

Inserisci il tuo codice identificativo, e premi il tasto

A994837774870

Invia la risposta

Figure 4: The form specifies a new feature, requested by an interested demander, not available in the apartment description. The owner can answer the question by clicking on the radiobutton and fill the textfield for the authentication procedure with his password.

terminological and assertional aspects of knowledge representation, and which focuses on the key inferences of subsumption and classification, typical of description logics.

Figure 1 shows the three-tier architecture of our prototype system. The web server passes requests to a Java servlet that communicates with the NeoClassic engine running as a daemon in the background. The main user interface for both supply and demand submission is a Java applet. Classic output is parsed in XML; the system is also interfaced to a mailer and to a SMS (Small Message System) server for automatic news notification.

Submission of a new query or offer is carried out incrementally. The user can introduce basic elements, such as price, type of the apartment, number of rooms, etc. A distinguishing aspect is the possibility to ask for new services or features not foreseen at the design stage. The system extends its knowledge and new concepts are added to the knowledge base. Figure 2 shows the form that includes a text area for specification of new concepts. The scrollable window in the upper part of the figure shows other concepts added and consequently part of the system knowledge. The system includes a small thesaurus and a predictive text input mechanism to avoid repeated insertion of similar concepts.

Our basic service is the matching of compatible descriptions. A query that matches a description will also subsume all other descriptions that in the system hierarchy are classified below the first matched description. Supplying a new apartment is also a description matching. The position of the new supply in the system hierarchy is determined considering all the descriptions that the new description is subsumed by. Once the position has been found, a reclassification takes place to determine the descriptions that satisfy it, which are then tied in the system hierarchy.

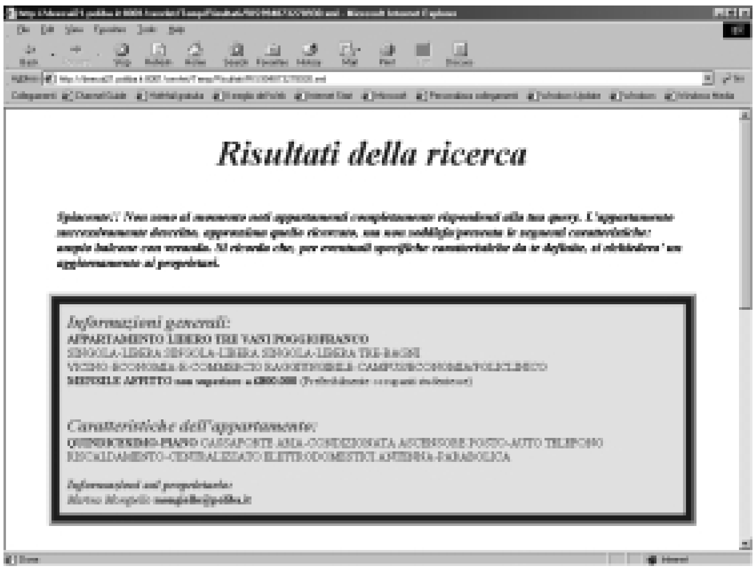
Each request, a supply or a demand, is given a unique code for information tracking. A supply is always an apartment, which can be rented as a whole or room by room or also on a "bed" basis. A demand can be any of the above. Test functions allow several deduction services including: reachable faculties given the address of an apartment and vice versa; computation of prices per room or per "bed" given the cost of an apartment; computation of spare rooms or spare beds. It is noteworthy that when a new concept is introduced by a user looking for a new service/feature, the system automatically contacts owners of apartments that fulfill all other user requirements but the new one through an e-mail or a SMS, see Figure 3 for an example.

The message points the owner to a web page where the question on the newly requested feature is posed, as it is shown in Figure 4. If he/she confirms the availability of the requested service/feature the system stores the new information and consequently informs, with the same procedure (an e-mail or a SMS), the demander about the fulfillment of his/her request.

The system also correctly handles requests that cannot be fulfilled for lack of apartments offers satisfying all needs expressed. The system answers declaring the reason why the query could not be completely satisfied, so that the user may eventually release the constraint, that is, turning to the hierarchical structure of the knowledge base, raise towards a less specified description.

It also proposes those apartments whose description, though not perfectly matched by the request are closer to it. We remark that this is the way we believe a human clerk would behave: he/she would propose something that, though not exactly in agreement with the customer request is close enough to be of interest. As an example Figure 5 shows the system answer, with the negative answer (and the reason for it) and alternative solutions.

If a user refuses alternatives his/her request is not lost. The system stores it; on insertion of new supplies matching the query the system automatically informs the user that a new offer is available satisfying his/her requests.



Risultati della ricerca

Spiacente!! Non sono al momento noti appartamenti completamente rispondenti alla tua query. L'appartamento successivamente descritto, approssima quello ricercato, ma non soddisfa/presenta le seguenti caratteristiche: ampio balcone con veranda. Si ricorda che, per eventuali specifiche caratteristiche da te definite, si chiederà un aggiornamento ai proprietari.

Informazioni generali:
APPARTAMENTO LIBERO TRE VANI POGGIOFRANCO
SINGOLA-LIBERA SINGOLA-LIBERA SINGOLA-LIBERA TRE-BAGNI
VICINO-ECONOMIA-E-COMMERCIO RAGGIUNGIBILE-CAMPUS/ECONOMIA/POLICLINICO
MENSILE AFFITTO non superiore a £800.000 ([illegible])

Caratteristiche dell'appartamento:
QUINDICESIMO-PIANO CASSAFORTE ARIA-CONDIZIONATA ASCENSORE POSTO-AUTO TELEFONO
RISCALDAMENTO-CENTRALIZZATO ELETTRODOMESTICI ANTENNA-PARABOLICA

Informazioni sul proprietario:
Marco Mongiello mongiello@poliba.it

Figure 5: Response to a query not completely matched. In the upper part the system provides information about the requirements not fulfilled. It points out that one requirement, "balcone con veranda" (i.e. porch), could not be fulfilled by any available supply. It also proposes in the lower part apartments that fulfill all other requirements of the query.

4 Conclusion and discussion

We have proposed the application of the Description Logics framework to supply-demand matching in a P2P scenario, modelling both as generic concepts to be matched. We have also presented as a realistic case study a prototype of system that implements a virtual brokering agency for apartments rental. The system is fully functional and is currently in alpha test stage.

Our conclusion is that Description Logics can be almost an ideal candidate for scenarios where peer entities interact for supply-demand matching, providing a level of interaction with user-information needs which is not available with current database technology. We end the paper discussing some pros and cons of our choices.

We had several reasons that led us to choose Classic instead of other implementations of Description Logics. First, since we are working on a years-long project, we need to rely on a stable system. Second, we need *system* functions like "list all roles attached to this concept", or "list the *derived* subsumers of this concept".

Moreover, we need a language escape for requirements that are not expressible in the Description Logic. Although weak as a *reasoner*, one of the strongholds of Classic is exactly the possibility to use computed fillers and test functions in the host language.

Third reason, but not less important, there is a large documentation on Classic, which is not just the system manual, but also tutorial-like papers [2] that allow newcomers to grasp very rapidly the system capabilities, and how to exploit them.

Nevertheless, we experienced also some problems in using Classic, such as the fact that computed fillers required almost continuous recompilation of the code. Also, Classic appears to have some strange behavior in the removal of "told" information, which requires a careful handling. Furthermore, lack of the disjunction operator (OR), and qualified existential quantification ("an apartment with a single room") required some work to simulate their behavior. We also needed small computing capabilities to express things as "an apartment with five rooms, three of which are rented, is an apartment with two free rooms".

To circumvent this problem in Classic, we had to code in concept names all combinations of available and rented rooms, and use rules; this appears a somewhat cumbersome compilation of subtraction.

Finally, we would have find useful some epistemic capabilities to express that, *e.g.*, to enter a new apartment in the system, its location must be entered too, or that the owner already told that smokers are allowed.

Acknowledgements

We wish to acknowledge help of our former students P. Bacco and M. Ranieri, and of P.F. Patel-Schneider who clarified various points of Neo-Classic implementation.

This work has been supported by project MURST-CLUSTER22 subcluster *"Monitoraggio ambiente e territorio"*, workpackage: *"Sistema informativo per il collocamento dei prodotti ortofrutticoli pugliesi"* and by EU-POP project *"Negotiation Agents for the Electronic Marktplace"*.

References

[1] Alexander Borgida. Description logics in data management. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):671–682, 1995.

[2] R. J. Brachman et al. Living with CLASSIC: When and how to use a KL-ONE-like language. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the representation of knowledge*, pages 401–456. Morgan Kaufmann, 1991.

[3] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proceedings of the Seventeenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[4] P. Devambu, S.G. Stubblebine, and M. Uschold. The next revolution: Free, Full, Open Person-2-Person (P2P) E-commerce. In *Proc. of DL 2000 Workshop*, 2000.

[5] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Reasoning in description logics. In Gerhard Brewka, editor, *Principles of Knowledge Representation*, Studies in Logic, Language and Information, pages 193–238. CSLI Publications, 1996.

[6] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[7] William A. Woods and James G. Schmolze. The KL-ONE family. In F. W. Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992. Published as a special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2–9.

Building a Text Adventure on Description Logic

Malte Gabsdil, Alexander Koller, Kristina Striegnitz
Dept. of Computational Linguistics
Saarland University, Saarbrücken, Germany
{gabsdil|koller|kris}@coli.uni-sb.de

Abstract

We describe an engine for a computer game which employs techniques from computational linguistics and theorem proving based on description logic. We show how we represent a world model as a DL knowledge base and then illustrate how we use it in the computational linguistics modules with the examples of analyzing and generating referring expressions.

1 Introduction

In this paper, we describe an engine for text adventures which employs techniques from computational linguistics and theorem proving based on description logic. The system is being developed at Saarland University as a student project. Its purpose is twofold: Players should be able to interact more naturally with the game, and we envisage a use as a testbed for computational linguistics modules.

Text adventures are a classical form of computer games which were most popular in the eighties. The player interacts with the game world (e.g. the rooms and objects in a space station) by typing natural-language commands and the computer provides feedback in the form of natural-language descriptions of the world and of the results of the player's actions. Typically, the user has to solve puzzles to win the game; an example interaction is shown in Fig. 1.

Text adventures have since gone somewhat out of fashion. One reason for this was the advent of more powerful graphics hardware, but another is that even the most advanced games of the eighties, which accepted input that went well beyond simple two-word sentences, suffered from some irritating limitations. Maybe most striking is what we call the *identification problem*: Sometimes the game does not allow the user to refer to an object with the exact same words that the game itself used for it (Fig. 2, taken from [3]). This is unsurprising, since the output of the game is hard-coded and elaborate, whereas the input has to be analyzed by a very simple parser.

```
Observation Lounge
This is where the station staff and visitors come to relax. There
are a lot of tables and chairs here, a large observation window,
and a plush carpet. In the corner you can see an AstroCola dispenser.
A tube leads up to the station's main corridor.

>put my galakmid coin into the dispenser
Click.
The dispenser display now reads "Credit = 1.00".

>push diet astrocola button
You hear a rumbling noise in the dispenser, but nothing appears in the
tray.

>kick dispenser
A can drops into the tray. Amazing! The oldest trick in the book, and
it actually worked.
```

Figure 1: An example interaction with a text adventure, taken from [7].

```
Cupboard
When you aren't lying on the bed, you usually stay in here, snug and
safe with your friends atop the warm pile of clothes. Your warm
winter jacket is here, which may be just as well, it's a little chilly.

>take the warm winter jacket
You can't see any such thing.

>take the winter jacket
You can't see any such thing.

>look at the jacket
A smart green jacket with big pockets, teddy bear sized.

>take the smart green jacket
You can't see any such thing.

>take the jacket with big pockets
I only understood you as far as wanting to take the green jacket.

>take the green jacket
Taken.
```

Figure 2: The identification problem.

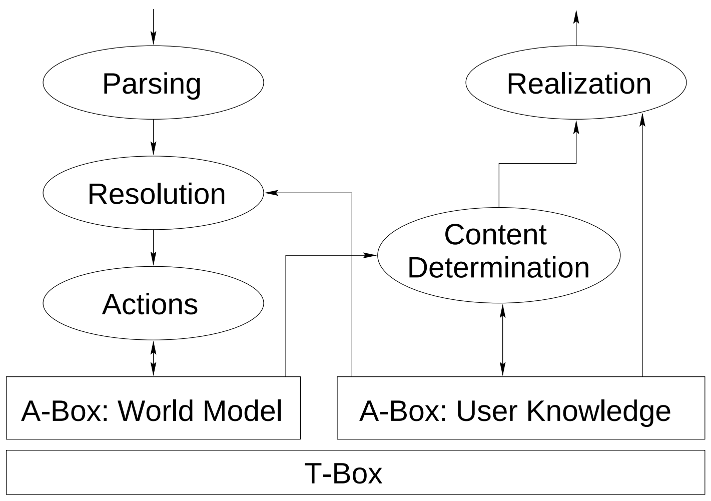


Figure 3: Architecture

Our system attempts to overcome this and other limitations by employing state-of-the-art techniques from computational linguistics, such as a real parser for English and a component for the automatic generation of the system's answers. Underlying the system is a world model based on description logic, which is used by almost every component of the NLP system. In our implementation we use the RACER system [5] because it provides support for A-Box reasoning, which is essential for us.

The paper is organized as follows: We will first sketch the general architecture of the system and its components (Section 2) and describe the DL world model (Section 3). Then we will briefly illustrate how we make use of DL inferences in the NLP modules by first showing how to analyze the meaning of referring expressions (Section 4), and then how to generate such referring expressions (Section 5). Section 6 concludes the paper and presents some ideas for future work.

2 Architecture

The general architecture of the game engine is shown in Fig. 3. The user's input is first parsed – that is, its syntactic structure is determined, using an efficient parser for dependency grammar [2]. Next, *referring expressions* (such as *the toolbox*) in the input are resolved to objects in the world. The result is a ground term that indicates the action the user wants to take.

This term is used to retrieve action descriptions from a database; the entry for "open" is shown in Fig. 4. Action descriptions are STRIPS-like operators defining preconditions and effects of the action. In addition, they specify in the 'uk' slot how the user knowledge has to be updated when the action is performed.

open(pat:X)	
pre:	closed(X), unlocked(X)
effects:	add: open(X) delete: closed(X)
uk:	add: open(X), describe(X) delete: closed(X)

Figure 4: The operator for the "open" action.

The term that was produced by the resolution component is matched with the head of the operator, binding the variables in the action description.

If the preconditions are satisfied, the world model is updated according to the 'effects' slot, and the instantiated contents of the 'uk' slot are passed on to the content determination component, which computes what information has to be verbalized by the generation module. This verbalization process is then carried out by a realization component based on Lexicalized Tree Adjoining Grammar [6, 10], which produces English text.

3 The World Model

The world model of the game engine is encoded as a DL knowledge base. The T-Box specifies the concepts and roles which are available in the world and defines complex concepts used e.g. by the resolution module (see below). The A-Boxes state which concepts and roles hold of the individuals in the world.

In the system, we use two different A-Boxes. One stores the current state of the world; it is used to determine whether the preconditions of an action are satisfied in the world, and, if this is the case, is updated with the action's effects. The function of the other A-Box is to keep track of the player's knowledge. It is used in the language-processing modules – for instance, referring expressions must be evaluated with respect to the user's knowledge –, and is updated by the content determination when it has determined what new information should be verbalized. The two A-Boxes share the same T-Box, but will typically be different. For instance, the world A-Box will usually contain more individuals than the user A-Box because the player will not have explored the world completely and will therefore not have seen all the individuals. On the other hand, it can be useful to deliberately hide effects of an action from the user, e.g. if pushing a button has an effect in a room that the player cannot see.

A fragment of the A-Box describing the state of the world is shown in Fig. 5. The T-Box specifies that the world is partitioned into three parts: rooms, objects, and players. The individual 'myself' is the only instance that we ever define of the concept 'player'. Individuals are connected to their locations (i.e. rooms,

room(scooter-bridge)	toolbox(t1)
hammer(h1)	player(myself)
saw(s1)	silver(t1)
closed(t1)	unlocked(t1)
has-location(t1, scooter-bridge)	has-location(h1, t1)
has-location(myself, scooter-bridge)	has-location(s1, t1)
...	

Figure 5: A fragment of the world A-Box.

container objects, or players) via the 'has-location' role; the A-Box also specifies what kind of object an individual is (e.g. 'toolbox') and what properties it has ('closed', 'silver'). The T-Box then contains axioms such as 'toolbox $\sqsubseteq$ object', 'silver $\sqsubseteq$ colour', etc., which establish a taxonomy among concepts.

These definitions allow us to add axioms to the T-Box which define more complex concepts. One is the concept 'here', which contains the room in which the player currently is – that is, every individual which can be reached over a has-location role from a player object.

$$\text{here} \doteq \exists \text{has-location}^{-1}.\text{player}$$

Another useful concept is 'accessible', which contains all individuals which the player can manipulate.

$$\text{accessible} \doteq \forall \text{has-location}.\text{here} \sqcup \forall \text{has-location}.(\text{accessible} \sqcap \text{open})$$

All objects in the same room as the player are accessible; if such an object is an open container, its contents are also accessible. As the player itself is by definition 'open', this includes the player's inventory.

Finally, we should mention that inside the action processing module, we create multiple temporary A-Boxes to allow for a more benevolent handling of ambiguity. Imagine the player types an ambiguous sentence, such as "put the apple in the box on the table". We will explain below how the resolution module can sometimes filter out some readings of such an ambiguity, but in this case, let's assume that it cannot decide whether the user meant putting "the apple" into "the box on the table", or "the apple in the box" onto "the table". It will hand both alternatives down to the action processing component.

Here we pursue all possible meanings of the sentence in parallel. For each reading, we create a copy of the current world A-Box, and then attempt to perform the action on the copy. This can be nontrivial because it may be possible to express sequences of actions with a single sentence, and these have to be performed one after another. If it turns out that we can only successfully perform the actions in one of the readings (e.g. because the player does not hold the apple in the box), we can commit to this reading without the user ever noticing that we had trouble understanding what he meant. Otherwise, we have to report an error.

4 Resolution of Referring Expressions

Referring expressions, such as *the toolbox*, *it*, or *a hammer*, link linguistic forms to objects in the world (the *referents* of the referring expressions). The player in our application will typically use definite descriptions (*the toolbox*) or pronouns (*it*) to refer to the objects on which he wants to perform an action. It is therefore essential to resolve these expressions to the actual individuals in the player-knowledge A-Box. As an example reconsider the A-Box in Fig. 5: We first have to resolve *the toolbox* to the RACER individual t1 before any action on this object can be carried out.

Resolving Definite Descriptions Definite descriptions of the form *the toolbox*, *the green apple*, or *the hammer in the toolbox* refer to an object that matches their restriction (*toolbox*, *green apple*, etc.). In a first approximation, we also take them to refer *uniquely*: That is, there must be exactly one object in the world that matches the restriction [9]. For *the toolbox*, this restriction is simply the concept 'toolbox'. We furthermore assume that the player will only try to refer to 'accessible' objects. This avoids confusion with other objects that would match the same description but are not in the same room as the player, i.e. objects that are not locationally salient. Thus we can retrieve a list of all potential referents for *the toolbox* by evoking the RACER query

$$\texttt{(concept-instances } \text{toolbox} \sqcap \text{accessible)}$$

Assuming that t1 is actually already present in the player A-Box, this returns the list `(t1)`. As it contains exactly one element, the reference succeeds; otherwise we would have rejected the command with an error message. Note that we always interpret reference with respect to the player's knowledge: The presence of toolboxes unknown to the player does not lead to an ambiguous reference.

More complicated definites are simply translated into more complex concepts. Our general strategy here is to push as much of the work into the DL inference problems and let RACER's optimizations work for us. For example, *the hammer in the toolbox* translates to the query

$$\texttt{(concept-instances } \text{hammer} \sqcap \text{accessible} \sqcap \exists\text{has-location.toolbox)}$$

The fact that definite references may fail (i.e. no referent in the world model can be found that matches the restriction) can be very helpful when we are faced with more than one possible syntactic derivation for an input. For example, the sentence *Unlock the toolbox with the key* is ambiguous: *The key* could either be an instrument used in the unlocking, or it could modify *the toolbox*, as in *the toolbox with the red handle.* In this example, we will not be able to find a referent for the constituent *the toolbox with the key* in the second parse, and will therefore only pass on the (resolved) first reading to the actions module.

Resolving Pronouns Unlike definite descriptions, pronouns do not provide much information about the object they refer to. However, the linguistic restrictions on which objects can be referred to by pronouns are much stricter.

We make use of a discourse model inspired by Strube's S-list [12] to determine the objects pronouns might refer to (their *antecedents*). The idea behind the S-list is to keep an ordered record of salient objects that have been introduced during a discourse and which are therefore most likely to be antecedents for pronouns. We associate every element in the S-list with agreement features (gender and number), its information status and text position (both needed to determine list-order; see [12]), as well as the RACER individual it refers to in the player A-Box. Resolving a pronoun then comes down to a lookup in the S-list: We simply take the first element that matches the pronoun's agreement restrictions.

The discourse model is updated incrementally and can be accessed by both the resolution and the generation module (see below). We can therefore resolve inter-sentential pronouns like *Take the apple and eat it* as well as simple cases of cross-speaker anaphora [4] as in the following short dialogue:

GAME:	*There is an apple on the table.*
PLAYER:	*Take it.*

5 Generation of Referring Expressions

The purpose of the generation module is to describe the environment the player is in and how his actions affect the game world. As is common practice in natural language generation systems, it consists of the two submodules *content determination* and *realization* (see e.g. [13]). Content determination assembles the information that has to be communicated to the player, and then passes it on to the realization module to cast it into a text.

The information that should be communicated to the user is essentially the value of the 'uk' slot of the instantiated action schema, with two notable differences. First, individuals can of course not be called by their internal names (such as t1), so we must again generate a referring expression that names them. Second, special atoms like 'describe(t1)' are taken as requests to generate a description of t1.

ds:	l1, l2
new:	l1:open(t1), l2:contains(t1,[h1,s1]), hammer(h1), saw(s1)
old instances:	t1

Figure 6: Output of Content Determination (Example: *Open the toolbox*)

Fig. 6 shows an example output of the content determination module for the action *open the toolbox*, as specified in Fig. 4. It tells the realization component to generate two sentences, called internally 'l1' and 'l2': one expresses the fact that the toolbox t1 is open now, and the other one introduces the objects that are contained in the toolbox. Depending on contextual factors, a possible output could be *The toolbox is now open. It contains a hammer and a saw.*

Referring to Objects Reference to *old* individuals, i.e. individuals the player already knows about, is mainly taken care of by the realization module (see below) as there is interaction with the surface form that is chosen for the referring expression.

In case the action schema asserts a fact about an individual r the player has *not* encountered before, this object will be introduced to the player by a *mini-description* stating the most obvious properties of the object – e.g., its (most specific) type, such as 'toolbox', and its colour. We can find out whether the individual is new to the player by checking whether it is an instance of the universal concept $\top$ in the player A-Box.

Suppose we want to generate a mini-description for the toolbox t1 in Fig. 5. The query

```
(individual-direct-types t1)
```

will return the list `(toolbox silver)`. Using concept subsumption checks, we can find out that 'toolbox' is the type of t1, and 'silver' is its colour; so both concepts go into the mini-description. (Subsumption checks are inexpensive because we can completely classify the T-Box when we start the system.) The realization component can use this information later to generate the expression like *a silver toolbox.*

Describing Objects There are two main situations in which object descriptions have to be produced. First, the player may ask explicitly for a description, for instance by saying *look at the toolbox.* In this case, a full detailed description of the object is required. Therefore, all (most specific) concepts of which the object is an individual as well as all role assertions in which the concept takes part are retrieved from the world model. Mini-descriptions (as above) are provided for the objects introduced through role assertions.

The second type of object description is intended for situations in which an action has changed the world in such a way that new information becomes

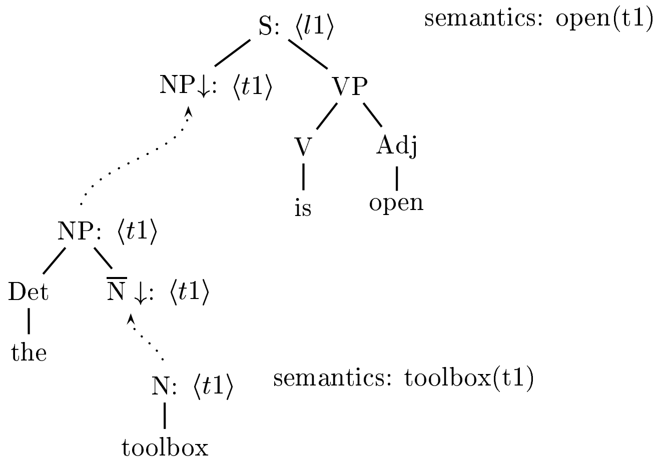


Figure 7: Realizing 'open(t1)'

accessible to the player. An example is the 'open' action, which makes the objects in the opened container visible. In this case, only new information, i.e. facts about the object that can be retrieved from the world model but don't follow from the user knowledge, should be contained in the description. At the moment, we derive this information from the specific action types. In the case of an 'open' action, not all role assertions are retrieved, but only those 'has-location' relations which point to objects included in the container. We aim to arrive at a more general solution eventually.

Realization The realization component produces a text to communicate the information assembled by the content determination to the player. In order for the text to be smooth and for the player to be able to correctly resolve references to objects, it is important that appropriate expressions are used for referring. For example, we want to refer to objects the player knows about with definite descriptions (*the toolbox*) and to new objects with indefinites (*a toolbox*). For new objects, we simply verbalize the concepts in the mini-description.

The correct verbalization of old objects is handled inside the main realization algorithm, which is based on [10]. In this framework realization comes down to assembling a sentence from the partial parse trees of a lexicalized tree-adjoining grammar [6]. Fig. 7 shows how *The toolbox is open* is built from fragments of syntax trees. The lexicon entries are associated not only with semantic information – which connects e.g. the word *toolbox* with the concept 'toolbox' –, but also with *pragmatic* information. This allows us to specify in the lexicon entry for *the* that definite descriptions must refer uniquely with respect to the player's knowledge. The resulting sentence (or little text) has to convey the information that content determination selected, it has to be syntactically viable (there must not be any holes in the result) and pragmatically appropriate.

To realize a reference to an old individual, we first compute the `individual-`

`direct-types` of the individual again; then we successively add the members of this list to the definite until the reference is unique, which we can check by computing the number of `concept-instances` as in Section 4. We follow standard practice in generation systems [1] by adding these concepts according to a predefined order of salience; first the type, then the colour, etc.

6 Conclusion and Outlook

In this paper, we have sketched the components of a text adventure engine which employs techniques from computational linguistics to make a more natural interaction with the game possible. The state of the world and the player knowledge are represented as description logic knowledge bases, and almost all language-processing modules utilize (A-Box and T-Box) inferences over these knowledge bases. We have looked more closely at the components for resolving and generating referring expressions, which solve the identification problem.

We are currently implementing the system; we hope to finish a prototype by September. The implementation is being done in the concurrent constraint programming language Oz [8], which allows us to reuse existing modules for parsing and realization [2, 11]. We communicate with the standalone version of RACER via sockets.

The system has much room for improvement, and indeed is designed in a modular fashion that will allow to replace specific components by more sophisticated ones. One line of future work could be to improve the part that resolves referring expressions; likewise, their generation is currently a very active research field in computational semantics, and new ideas could easily be incorporated into the system. One straightforward improvement of the realization component would be to add lexical entries that contain larger chunks of text, which could make the output more interesting to read.

Beyond these local changes, one can imagine many additions to the system's functionality. For instance, one could add speech recognition and generation components. In addition, it would be interesting to allow multiple instances of 'player' and make the game multi-user, or to model the world more realistically e.g. by replacing the room concept by coordinates in the world. But we believe that even the first version as it stands offers an interesting setup for exploring the use of description logic in computational linguistics.

Acknowledgments. We are grateful first of all to our students, without whose enthusiasm in implementing the system the game would have remained an idea. We are indebted to Ralph Debusmann for his contributions to the parsing component and the syntax-semantics interface. Carlos Areces introduced us to the new world of efficient DL provers, and Volker Haarslev and Ralf Möller were wonderfully responsive in providing technical support for RACER. Special thanks go

to Gerd Fliedner, in a discussion with whom the idea for employing techniques of computational linguistics in a text adventure engine came up first.

References

[1] Robert Dale and Ehud Reiter. Computational interpretations of the gricean maxims in the generation of referring expressions. *Cognitive Science*, 18:233–263, 1995.

[2] Denys Duchier and Ralph Debusmann. Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings of the 39th ACL*, Toulouse, France, 2001.

[3] David Dyte. A Bear's Night Out. Text adventure. Available at `http://www.covehurst.net/ddyte/abno/`, 1997.

[4] Nissim Francez and Jonathan Berg. A Multi-Agent Extension of DRT. In H. Bunt, R. Muskens, and G. Rentier, editors, *Proceeding of the* 1^{st} *International Workshop on Computational Semantics*, pages 81–90, 1994.

[5] Volker Haarslev and Ralf Möller. RACER System Description. In *Proceedings of IJCAR-01*, Siena, 2001.

[6] Aravind Joshi and Yves Schabes. Tree-Adjoining Grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, chapter 2, pages 69–123. Springer-Verlag, Berlin, 1997.

[7] David Ledgard. Space Station. Text adventure, modelled after a sample transcript of Infocom's *Planetfall* game. Available at `http://members.tripod.com/~infoscripts/planetfa.htm`, 1999.

[8] Mozart Consortium. The Mozart Programming System web pages. `http://www.mozart-oz.org/`, 1999.

[9] Bertrand Russell. On Denoting. *Mind*, 14:479–493, 1905.

[10] Matthew Stone and Christine Doran. Sentence planning as description using tree adjoining grammar. In *Proceedings of ACL*, pages 198–205, 1997.

[11] Kristina Striegnitz. Model Checking for Contextual Reasoning in NLG. In P. Blackburn and M. Kohlhase, editors, *Proceedings of ICOS-3*, Siena, 2001.

[12] Michael Strube. Never Look Back: An Alternative to Centering. In *COLING-ACL*, pages 1251–1257, 1998.

[13] H.S. Thompson. Strategy and Tactics in language production. In *Papers from the 13th Regional Meeting of the Chicago Linguistic Society.* 1977.

Description Logics for Matchmaking of Services.

Javier González-Castillo, David Trastour*and Claudio Bartolini
HP-Labs, Bristol, UK
Email: javgon@hplb.hpl.hp.com, david_trastour@hp.com,
claudio_bartolini@hp.com

Abstract

Matchmaking is an important aspect of E-Commerce interactions. The current trend in B2B E-Commerce automation is towards complex interactions for service provision. In this context, matchmaking services require rich and flexible metadata as well as matching algorithms. The Semantic Web initiative at W3C is gaining momentum and generating suitable technologies and tools to cover both the metadata and the algorithmic aspects. In this paper we describe our experience in building a matchmaking prototype. We choose to base our prototype on a Description Logic (DL) reasoner, operating on service descriptions in DAML+OIL. We report on our investigation of DAML+OIL to express service descriptions and on our experience on existing DL reasoners, in particular assessing RACER and FACT.

1 Introduction

The automation side of E-Commerce transactions brings many advantages to businesses in dealing with their partners, customers, and suppliers. The increased efficiency and fewer errors in computations make possible higher throughput and further reach, and therefore open up the possibility of interacting with a far greater number of potential counterparts. But with the enlarged possibilities comes the problem of having to select the best among the multitude of available counterparts. Such selection must happen based on various aspects of the business offers that providers make available and requestors seek for. Matchmaking is the process of pruning the space of possible matches among compatible offers and requests.

*Please consider David Trastour as the main contact.

There are two factors that play in making matchmaking in B2B E-Commerce a difficult problem. On one hand, service provision interactions evolve to be ever more complex. This requires that the language for service descriptions for matchmaking be expressive enough to deal with this complexity. On the other hand the sheer number of potential solutions heavily constrains the efficiency that is achievable. This, united with a requirement for accuracy in reporting matching offers and requests, makes the problem barely tractable with traditional techniques.

Because Semantic Web technologies promise to transform the information on the web from human-readable to machine-understandable [14], we think that the application of these technologies may be valuable to our aim. In particular, we have studied DAML+OIL as we believe that a subset of it could be used to describe service parameters. Because DAML+OIL is heavily influenced by Description Logics, it seems natural to use a DL reasoner as the heart of the engine that calculates the matches.

The remainder of this paper is structured as follows. Section 2 describes matchmaking in detail. In section 3 we analyze how DL could be a solution for matchmaking and we describe a matching algorithm based on the subsumption tree given by a DL reasoner. In section 4 we describe our experience with different DL reasoners and list some requirements we would like to see for future DL reasoners. Section 5 talks about future work and we conclude in section 6.

2 Matchmaker

With the proliferation of offers comes the problem of finding and selecting potential counterparts for service provision/consumption. The sole presence of many potential buyers and sellers on the web is not a sufficient condition for them doing business together. Through the mediation of the matchmaker, which matches service offers with service requests, potential counterparts will be able to find each other.

2.1 Service Description Language

Service description is a very broad term subject to different interpretations. For example, WSDL (Web Service Description Language) [8] descriptions focus on the behavioral aspects of a service. UDDI (Universal Descripition, Discovery and integration of Business for the Web) [5] descriptions are based on three different types of information: contacting details – white pages –, classification with respect to a certain taxonomy – yellow pages –, and technical information – green pages.

The purpose of our work is to embrace and extend Web Services descriptions, taking a more general approach while providing the expressiveness and flexibility that we require. Our approach is based on expressing service descriptions through a set of complex parameters. These parameters are used to express a variety of aspects of the service and the entities involved. Our framework must be flexible enough to accommodate descriptions with various levels of complexity, from the simple sale of a good to a complex business interaction.

2.1.1 Requirements

Previous investigations [17] on the application of RDF/RDFS [14, 6] to service description in the context of matchmaking lead us to he following requirements:

- **High degree of flexibility and expressiveness.** The advertiser must have total freedom to compose the service description. Different advertisers will want to describe their services with different degrees of complexity and completeness, and our language must be adaptable to these needs. An advertisement may be very descriptive in some points, but leave others less specified and open for negotiation a posteriori. Therefore, ability to express semi-structured data is required.

- **Support for Types and Subsumption.** We do not want to restrict matching to be based on simple string comparison. A type system with subsumption relationships is required, so more complex matches can be provided based in these relationships.

- **Support for Datatypes.** Attributes such as quantities, prices, or dates will be part of the service descriptions. The best way to express and compare this information is by means of datatypes. As a starting point, we will deal with datatypes such as `real`, `date`, `string` etc.

- **Express Restrictions and Constrains.** Whether it is an offer or a request, it is often the case that what is expressed is not a single instance of a service but rather a conceptual definition of the acceptable instances. A natural way of describing this is by expressing constraints over the parameters of the service.

- **Semantic level of Agreement.** In order to compare descriptions, they need to share the same semantics.

- **Appropriate syntax for the Web.** The matchmaker must be compatible with Web technologies and the information must be in a format appropriate for the Web.

2.1.2 DAML Approach

DAML+OIL is one of the most promising technology of the Semantic Web activity. This ontology mark-up language for web resources developed by DARPA provides a richer set of modeling primitives than other ontology languages such as RDF/RDFS. In its last specification [18] it has been extended with arbitrary datatypes from XML Schema [4].

To fulfill our requirements, we are proposing to represent the concepts in a service description as DAML+OIL classes. The service description is defined as the boolean combination of a set of restrictions over datatype and abstract properties. These restrictions are expressed either through DAML+OIL restrictions or XML Schema restrictions. It is worth noting that service description ontologies and domain-specific ontologies also have an important role to play in order to achieve the semantic level of agreement between the various parties. The example service description ontology we have developed uses the class `srcv:ServiceDescription` to denote the root of a service description.

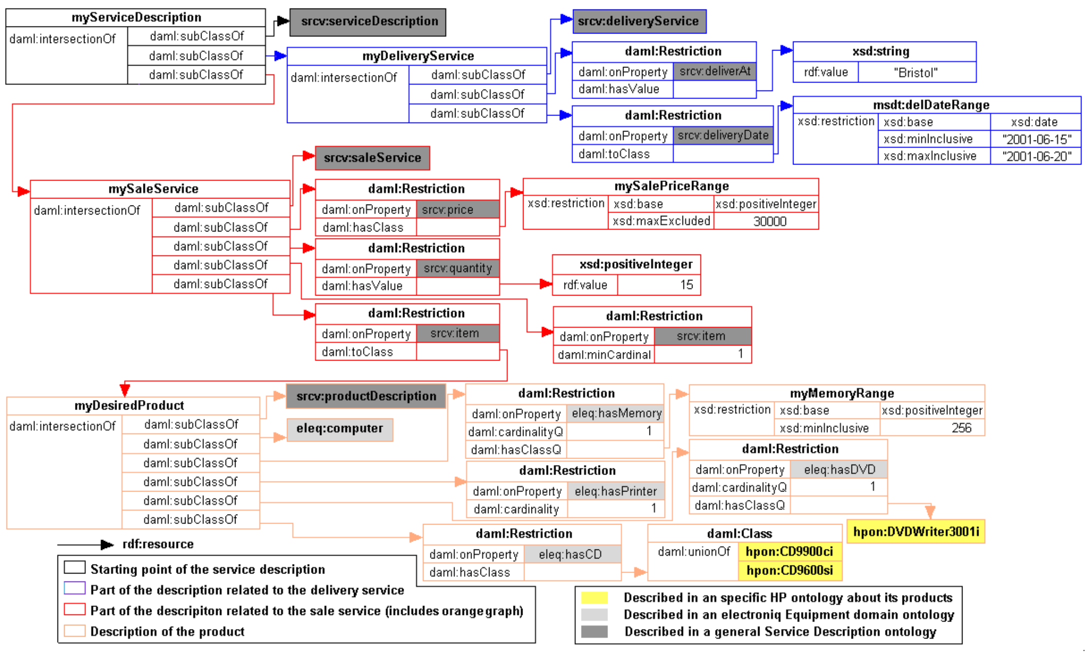


Figure 1: Example: Description of a service

Let us consider an example of a composite service of sale and delivery of computers. For the sole purpose of this example, we have defined a service description ontology and some electronic equipment related ontologies. We want to emphasis on how DAML+OIL is used to describe services requests and offers. Figure 1 represents the service of sale of 15 items. These items must be of type `eleq:computer` and must satisfy the following constraints: has at least 256 MB of RAM memory; has a printer – any make –; has a DVD unit, model HP DVDWriter 3001i; and has a CD unit, HP-CD9900ci or HP-CD9600si. There is an additional restriction on this service of sale stating that the price must be less than 30,000. The service of delivery has the following constraints: the goods must be addressed to "Bristol" any day between 15-06-2001 and 20-06-2001 included.

In order to construct this description we make use of different classes and properties. Some of them have been previously defined in some domain specific ontologies written by third parties - boxes filled with different colors -, while others such as `myServiceDescription`, `mySaleService`, `myDeliveryService` and `myDesiredProduct` are defined here in terms of restrictions over properties. All boxes in the figure named `daml:Restriction` represent these restrictions. Depending on the slots inside these boxes, we can distingish different types of restrictions such as: existential restrictions, value restrictions or cardinality restrictions.

2.2 Concept of match

Different approaches to matching can be taken. Existing solutions like UDDI or ebXML[1] (Electronic Business XML) manage to give accurate results at the expense of expressiveness by having a rigid format for descriptions and by restricting the query mechanism. Based on real-life examples like yellow pages directories, advertisement newspapers or bulletin boards, we would rather be able to compare descriptions with different levels of specificity and complexity than use an approach based on exact matching. For instance a *general* description for the sale of PCs, without any restrictions, should match the above example. More *specific* descriptions should also be matched. Finally, descriptions that are neither more specific or more general but that describe services that would be compatible with our example should also match.

Definition 1 *A service description is a self-consistent collection of restrictions over named properties of a service.*

Definition 2 *A service description D1 is a match for a service description D2 if there is no contradiction between all of the restrictions in D1 and D2.*

In the following section, we look at how DL reasoners can help us find matches among DAML+OIL based service descriptions.

[1]Suite of specifications that enables enterprises to conduct business over the Internet.

3 Description Logics

Description Logics are a family of knowledge representation formalisms. They are based on the notion of concepts and roles, and are mainly characterised by constructors that allow complex concepts and roles to be built from atomic ones [11]. The main benefit from these knowledge languages is that sound and complete algorithms for the subsumption and satisfiability problems often exist. A DL reasoner solves the problems of equivalence, satisfiability and subsumption.

3.1 DL and DAML+OIL

Because DAML+OIL has been influenced by DL, it appears natural to apply DL techniques to classsify our service descriptions.

As we will see in the following section, none of the DL variants for which there exists an implementation of a reasoner possesses enough expressiveness to deal with the whole set of constructors that form the DAML+OIL language. If we want to adopt a DL solution for implementing the matchmaker, we must restrict the descriptions to a subset of DAML+OIL.

At the moment, the most advanced available reasoners are for the $\mathcal{SHIQ}$ DL. This DL supports most of the DAML+OIL language, but its main drawback is that it cannot deal with individuals or datatypes in the definition of concepts. From our list of requirements this is too restrictive.

$\mathcal{SHOQ}$(D) is more expressive than $\mathcal{SHIQ}$ as it adds individuals and datatypes support – even though it does not allow for inverse role[2]. Not having the inverse role property does not cause us any major concern while individuals and concrete datatypes are quite essential to our application. A complete algorithm for solving the subsumption and satisfiability problem for $\mathcal{SHOQ}$(D) exists [12], but we do not know of any implementation available.

3.2 DL for Matchmaking

In this section we are describing the functionalities of a matchmaking service and a DL based matching algorithm needed to provide them.

3.2.1 Matchmaker functionalities

Our matchmaking service provides three basic functionalities [17]:

Advertising is the act of publishing a service description, or advertisement, to a matchmaking service. Before an advertisement is included in the

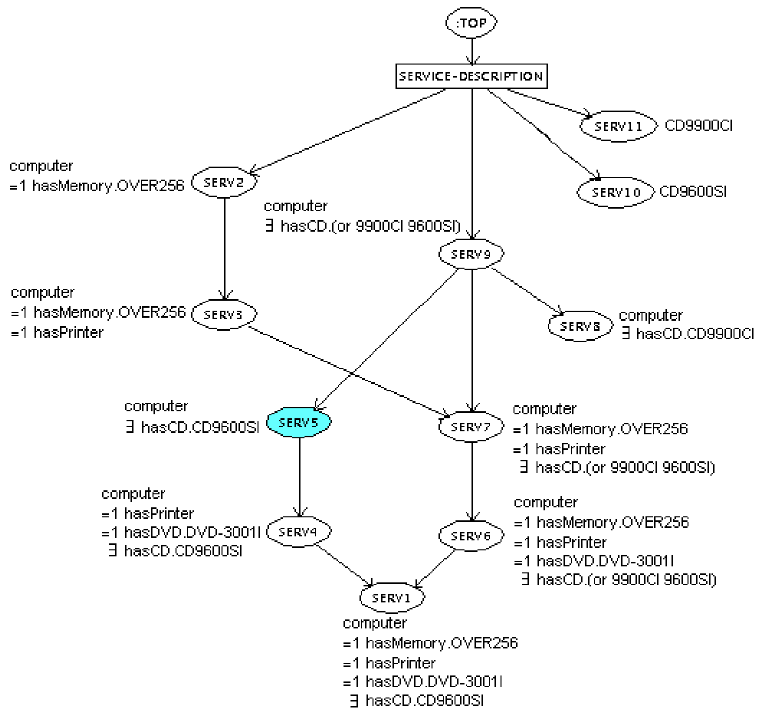
[2]Reasoning with both concrete domains or individuals plus inverse roles is known to be difficult and/or highly intractable [12].

knowledge base of the matchmaker, the satisfiability of all its concepts must checked because the realization of a non satisfiable service is not possible. When accepted, an advertisement becomes a set of new concepts within the subsumption tree. One of these concepts, the one under the *serviceDescription* branch, represents the whole advertisement.

Querying is similar to advertising except that the description submitted to the matchmaker is not persistent. The algorithm in the next section let us calculate the matches with a DL reasoner.

Browsing allows parties to find out about published advertisements. Browsing parties can make use of this information to tune the advertisement or queries that they submit in turn, so as to maximize the likelihood of matching. Browsing is based on navigating the subsumption tree through the branches provided by our service description ontology.

3.2.2 Matchmaking Algorithm



:TOP
SERVICE-DESCRIPTION
SERV11 CD9900CI
SERV10 CD9600SI
computer =1 hasMemory.OVER256 SERV2
computer ∃ hasCD.(or 9900CI 9600SI)
SERV9
computer =1 hasMemory.OVER256 =1 hasPrinter SERV3
SERV8 computer ∃ hasCD.CD9900CI
computer ∃ hasCD.CD9600SI SERV5
SERV7 computer =1 hasMemory.OVER256 =1 hasPrinter ∃ hasCD.(or 9900CI 9600SI)
computer =1 hasPrinter =1 hasDVD.DVD-3001I ∃ hasCD.CD9600SI SERV4
SERV6 computer =1 hasMemory.OVER256 =1 hasPrinter =1 hasDVD.DVD-3001I ∃ hasCD.(or 9900CI 9600SI)
SERV1 computer =1 hasMemory.OVER256 =1 hasPrinter =1 hasDVD.DVD-3001I ∃ hasCD.CD9600SI

Figure 2: Service description branch of the subsumption tree

Definition 3 *The matches for service description S are:*

- *equivalent concepts to S;*
- *sub-concepts of S;*
- *super-concepts of S that are subsumed by the serviceDescription concept;*
- *sub-concepts of any direct super-concept of S whose intersection with S is satisfiable.*

The algorithm is a translation in DL terms of the ideas exposed in the previous section.

To understand the algorithm in more detail, we are applying it to the example depicted in Figure 2. This figure shows the *serviceDescription* branch of the subsumption tree in the matchmaker at the moment of the query. Nine advertisements of sale and delivery of computers and two of sale of CD units have been published. We are considering a party interested in finding a computer which has a CD unit CD9600Si. Her query is denoted as SERV5 in the figure (filled node).

We evaluate sequentially the four propositions of the algorithm. In our example, there is no equivalent concept to SERV5. SERV4 and SERV1 are sub-concepts of SERV5 and as such are marked as matches. The third step is to look for super-concepts of SERV5 – up to the *serviceDescription* node. Hence, SERV9 is marked as a match. Finally, the last step of the algorithm gives us the nodes SERV6 and SERV7. While these nodes are neither super or sub concepts, they are compatible to the SERV5 query[3], in that the restrictions over the properties that appear in them and SERV5 are not inconsistent.

The problem of presenting results back to the user in a way that make sense to her (i.e. any ordering based on preferences) is beyond the scope of our current work.

4 Practical Approach

In this section we report on our experience on existing DL reasoners, in particular assessing RACER and FACT. We also list a set of requirements for future DL reasoner suitable for our application.

4.1 Experiences with existing DL Reasoners

4.1.1 FaCT Reasoner

The FaCT [13] system is a DL classifier being developed by Ian Horrocks from the Department of Computer Science at the University of Manchester. It includes

[3]Even though SERV8 is a sub-concept of SERV9 – which is a match –, it is not given as result because it contradicts the SERV5 query.

two reasoners for TBoxes, one of them for the $\mathcal{SHIQ}$ logic. Therefore, it cannot deal with individuals or concrete datatype domains, and a description such as the one in Figure 1 can not be processed with it.

To cope with the limitation of $\mathcal{SHIQ}$, we tried to model nominals, datatypes, and datatype values as atomic concepts but this can lead to incorrect inferences [12], not to mention the need to model one atomic concept for each integer.

DAML+OIL uses namespaces and import statements to provide extensibility and to deal with the distributed nature of the Web. The support in the reasoner for multiple interconnected TBoxes would solve this problem as we would model each DAML+OIL ontology in a different TBox. Because FaCT does not support multiple TBoxes we are using fully qualified names in a single TBox.

Moreover, the knowledge base of the matchmaker will change over time by addition of new advertisements as well as deletion or modification of existing ones. FaCT deals with the addition of new classes over time, even after classification has been done, but doesn't provide a mechanism for removing classes in the classification. This is a requirement for our application.

One of the main benefits of this system is its CORBA interface [3] that makes the reasoner available as a service for other applications to use. It also provides XML syntax for the definition of ontologies. To load our descriptions in the reasoner, we are translating DAML+OIL descriptions to the FaCT XML syntax.

4.1.2 RACER Reasoner

RACER [9, 10] is the first reasoner for TBox and ABox for the $\mathcal{SHIQ}$ logic. It is developed at the Computer Science Department of the University of Hamburg.

Like FaCT, it only provides part of the expressiveness that we need for our application. It is able to deal with multiple TBoxes, but they are not interconnected. It does not let us define a concept in a TBox in terms of concepts or roles from other Tboxes.

RACER does not provide support for a dynamic knowledge base as it is not possible to add or remove concepts once the classification has been done.

Another interesting feature of RACER is its ability to reason about ABoxes. With our approach to matchmaking this capability is not strictly necessary, as we only need to reason about concepts, for which TBoxes provide the necessary abstraction. However, the ability to reason about ABoxes may prove useful when extending our framework to cover phases of E-Commerce transaction beyond matchmaking. For example, an agreement struck between two parties following matchmaking and automated negotiation [2], needs full instantiation of the parameters that originally appeared in the service descriptions. Support for ABoxes would enable compliance check of the agreement with the negotiation proposals and with the original service descriptions in turn.

RACER provides a Java API and allows access to the reasoner remotely.

4.2 Requirements for a DL reasoner for matchmaking

From our experience, we have gathered the following requirements for a DL reasoner that would satisfy our needs:

- $\mathcal{SHOQ}(\mathbf{D})$ is the minimum expressiveness required;
- **Dynamic.** Advertisements will be added, removed and modified and the concepts within the knowledge base will need to be re-classified.
- **Ability to deal with multiple interconnected Tboxes.** We want to use different ontologies, and define concepts based on other external concepts and roles.
- **Scalability.** The reasoner needs to be able to cope with large amounts of information in an efficient way.
- **Persistency.** Storage of the advertisements is needed. The reasoner needs to be integrated with some form of persistent store in a way that maintains data consistency.
- **Support for DAML+OIL syntax** would avoid unnecessary translations.

5 Future Work

Our two prototype implementations of a matchmaker are fairly similar in terms of functionalities and are both incomplete. To go further into the development, we are lacking a DL reasoner with the properties mentioned above. The main requirement would be the support for the right level of expressiveness: $\mathcal{SHOQ}$(D).

On the service description side, we realize that the model we are proposing restricts the description of the service to a set of parameters. While this approach fits well with simple services like catalog-based solutions for the sale of goods, we recognize the need for a behavioural description for complex services. While all the examples of this paper only exposed buyer-seller relationships, we need to envisage a world where parties want to interact though complex business processes. The matchmaking of potential counterparts would then need to consider not only the service parameters, but also the compatibility between the various roles and behaviours. We want to include this work with another activity we are carring out on cooperative business processes [15].

Recently the DAML community has announced and release a first version of DAML-S, the Web Service Mark-up Language [7]. DAML-S is a Web service ontology which will allow software agents to discover, invoke, compose and monitor the execution of Web Services. To validate our ideas we have developed a primitive service description ontology. Our work could only benefit from using a full-fledged service description ontology. We will try to leverage from DAML-S as much as we can.

We are tracking what the Semantic Web community is producing in terms of tools but more specifically persistent stores for RDF and DAML. In particular we find the work on RQL [1] very promising. We need to envisage how to integrate DL reasoners with a persistent store.

6 Conclusions

Our experience in prototyping a DL based matchmaking service made us realize that there is a gap between what standard technologies for E-Commerce provide today and what could be achieved through the use of Semantic Web technologies. We believe that in the near future automated matchmaking and negotiation will achieve results at a level of complexity far beyond what is possible today. In particular the use DAML+OIL and of the evolution of DL reasoners like FaCT or RACER will play a primary role in making that happen.

References

[1] V. Alexaki; et al. *The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases, Proceedings of the Second International Workshop on the Semantic Web.* SemWeb'2001. May 2001.

[2] C. Bartolini and C. Preist *A Framework for Automated Negotiation.* 2001. HP Labs Technical Report.

[3] S. Bechhofer, I. Horrocks and S. Tessaris. *CORBA interface for a DL Classifier.* March 1999.

[4] P.V. Biron, A. Malhotra. *XML Schema Part 2: Datatypes.* W3C Recommendation 02 May 2001.

[5] T. Boubez; et al. *UDDI Data Structure Reference V1.0.* September 2000.

[6] D. Brickley and R.V. Guha. *Resource Description Framework (RDF) Schema Specification 1.0.* W3C Candidate Recommendation 27 March 2000.

[7] M. Burstein et al. *DAML-S: Semantic Markup for Web Services.* Part of the DAML-S Draft Release (May 2001).

[8] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana. *Web Services Description Langauge (WSDL) 1.1.* January 2001.

[9] V. Haarslev and R. *Möller. RACER User's Guide and Reference Manual Version 1.5.6.* April 2001.

[10] V. Haarslev and R. *Möller. RACER System Description.* To appear in: International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, 2001, Siena, Italy.

[11] I. Horrocks, U.Sattler and S. Tobies. *Practical reasoning for expressive description logics.*In H. Ganzinger, D. McAllester, and A. Voronkov, editors, Proceedings of LPAR'99, vol. 1705 of LNAI, pages 161–180. Springer, 1999.

[12] I. Horrocks. *Ontology Reasoning for the Semantic Web.* Network Inference 2001.

[13] I. Horrocks. *FaCT Reference Manual Version 1.6.* August 1998.

[14] O. Lassila and R. Swick. *Resource Description Framework (RDF) Model and Syntax specification.* W3C Recommendation 22 February 1999.

[15] G. Picinelli and L. Mokrushin. *Dynamic Service Aggregation in Electronic Marketplaces.* 2001. HP Labs Technical Report.

[16] U. Sattler. *A concept language extended with different kinds of transitive roles.* In 20. Deutsche Jahrestagung *für* KI, LNAI 1137.

[17] D. Trastour, C. Bartolini and J. Gonzalez. *A semantic Web Approach to Service Description for Matchmaking of Services.* Semantic Web Workshop Symposium 2001. To appear.

[18] F. van Harmelen, P.F. Patel-Schneider and I. Horrocks *Reference description of the DAML+OIL (March 2001) ontology markup language.* March 2001.

Modeling X.509 Certificate Policies Using Description Logics

Stephan Grill
Institute for Applied Information Processing and Communications
Graz University of Technology, Austria
Email: stephan.grill@plusultra.at

Abstract

Public Key Infrastructures are gaining importance in today's IT environment for managing certificates and keys. It is recognized, that the quality and trustworthiness of certificates depend to a large extend on the practices and procedures a certification authority applies when issuing certificates. These procedures are documented in certificate policies, which are generally text-based documents and therefore cannot be processed by machines. This paper describes a framework based on description logics that addresses this situation. Subsumption will be used to compare policies. Based on a case study of modeling real policies some features of this framework will be described. Learnings and an outlook of future work conclude this paper.

1 Introduction

Public Key Infrastructures (PKI) are emerging as an important cornerstone of today's communications systems. They are envisioned to enable a wealth of services ranging from electronic id cards, digital signature, authorization schemes, etc. and are already increasingly used in web-applications, e-mail, e-payment.

Processes and protocols to manage and use private keys and certificates are well understood and corresponding standards [1] are currently in the process of being defined.

The basic concept of a PKI is that participants use key pairs consisting of a private and a public key. The private key never leaves the trusted environment of the user and might for example be used for signing a document, whereas the public key is published for others to verify the signature created with the corresponding private key. To associate a person with a public key a trusted third party issues certificates, which express this association.

A trusted third party (also called certification authority CA) can issue certificates according to different policies. Policies define practices followed by the CA in authenticating the subject, the users obligations in protecting the private key, the legal obligations of a CA, etc. Different policies represent different security levels. For example, a policy might state that certificates are issued for public keys whose corresponding private keys are generated and stored on a smart card and a certificate requestor has to be authenticated in person using an id card, whereas a different policy might state that the private key is stored on a PC and only the existence of the e-mail address of the requestor has been verified. Obviously, information contained in a certificate issued according to the first policy is much more reliable and the certificate is therefore more secure than a certificate issued according to the latter policy. Consequently, the trustworthiness of signatures depends on the security level of the associated certificate and users verifying signatures have to make sure they are aware of the associated policies.

However, these policies are currently represented in textual form and therefore hardly ever inspected! This paper describes how description logics can be applied to represent certificate policies in a structured manner and to make them comparable.

2 Requirements for the Representation of a Policy

To define requirements for a policy representation, cases for using these models will be discussed first:

- Users should be enabled to *retrieve* specific parts of a certificate policy (e.g. they want to check where the associated private key of a certificate is generated and stored)
- Users might want to *specify properties* a certificate policy must match in order to be accepted (e.g. they want to specify that only certificates are accepted, if the associated private key is generated and stored on a smart-card).
- These comparisons might be based on equality comparisons but also other *comparison operators* must be supported which might be applicable for the quality of certificates (a policy might indicate that associated certificates are more trustworthy than certificates issues according to another policy).

From these possible use cases a set of requirements can be derived:

- Current text-based policies describe complex objects and complex practices. A formal model must support this complexity. The representation has to support <*attribute*, *value*> pairs, which might be organized in a hierarchical structure, and where *value* by itself might be a complex object.
- The representation must allow to define a metric and/or classification scheme to support not just equality comparisons, but also additional relational operators.

– The representation must be declarative (opposed to a procedural representation) in order to support operations on this representation.

3 Proposed Solution

A two-phased approach was chosen to address this problem. In the first phase a possible semantic representation is investigated, and in the second phase the syntactical representation of the defined semantics is defined.

Description logic was chosen for the semantic model because it provides an expressive data model and the required operators [3].

Currently an investigation is ongoing to identify a suitable syntactic representation. Likely candidates are the Resource Description Framework RDF [5] of the W3C or DAML+OIL [9].

Certificate policies are modeled in process consisting of three-steps:

1. definition of the semantics of the taxonomy
2. definition of a reference ontology
3. definition of individual policies

3.1 Definition of the Semantics of the Taxonomy

The objective of using a classification based knowledge representation mechanism is to automatically induce an order relationship in the concept space. For modeling policies, the taxonomy is used to represent information related to security aspects:

if concept C1 subsumes concept C2 then C1 is less secure than C2 – i.e. concepts higher up in the concept taxonomy will be considered less secure than concepts further down in the hierarchy

This decision is basically an arbitrary, but a meaningful one. Concepts higher in the taxonomy are less detailed specified as concepts further down. From a security point of view concepts with a more detailed specification are preferred to concepts with a less detailed specification. This is the case because the more information is given about a concept the less ambiguity is possible.

Other definitions are possible as well; however it is not allowed to mix taxonomies, which represent different semantics, within a specification of a single policy.

3.2 Definition of a Reference Ontology

The objective of defining a reference ontology is to

– define a *common* and re-usable *terminology*,

– create definitions that make individual policies *comparable*
– optionally define a *primitive taxonomy* explicitly

It defines concepts and individuals, which can be refined and combined by subsequent policy descriptions using specialization and the definition of new concepts respectively. These concepts and individuals define a core terminology, which ought to be accepted as a common framework. It will be possible that several of such reference frameworks will be developed and these frameworks can be combined.

3.3 Definition of an Individual Policy

Finally concepts will be defined to create certificate policies. These concepts use the reference ontology and therefore are based on the order relationship created through the definition of the taxonomy of the reference ontology.

4 Case Study

To verify the applicability of this approach a reference ontology using the framework defined in RFC 2527 [2] and two actual policies have been modeled using the DL-system NEOCLASSIC [4]. The chosen policies are

– Certificate Policies for the Government of Canada (GoC) Public Key Infrastructure [6]
– Swedish SEIS Certificate Policy [7]

The GoC PKI Policies actually define policies representing different assurance levels: rudimentary (1), basic (2), medium (3), high (4). In the following examples policies for signature certificates will be identified with the prefixes `GocSign[1234]`.

The reason for choosing these policies was, that all are based on the framework suggested in RFC 2527 - and therefore provide a possibility to make them comparable.

The following examples will show this approach more specifically.

4.1 Key Sizes

The first example is rather simple; it is using the predefined properties of the build-in concept `Integer`.

The core terminology based on RFC 2527 requires describing the minimal length of the used keys:

```
( createConcept Rfc2527AsymmetricKeySizes
    ( all keyLength Integer ) )
```

Concept `Rfc2527AsymmetricKeySizes` has one role named `keyLength`, which is of type `Integer` (a built-in NEOCLASSIC concept).

The Government of Canada PKI policy defines the following restrictions on the above concept:

```
( createConcept GocSign1AsymmetricKeySizes
    ( and Rfc2527AsymmetricKeySizes
          ( all keyLength ( minimum  512 ) ) ) )
```

Concept `GocSign1AsymmetricKeySizes` is a sub-class of `Rfc2527Asym-metricKeySizes` with the additional restrictions that all values of the attribute `keyLength` must be greater-equal 512. Similar restrictions apply for the remaining assurance levels:

```
( createConcept GocSign2AsymmetricKeySizes
    ( and Rfc2527AsymmetricKeySizes
          ( all keyLength ( minimum 1024 ) ) ) )

( createConcept GocSign3AsymmetricKeySizes
    ( and Rfc2527AsymmetricKeySizes
          ( all keyLength ( minimum 1024 ) ) ) )

( createConcept GocSign4AsymmetricKeySizes
    ( and Rfc2527AsymmetricKeySizes
          ( all keyLength ( minimum 2048 ) ) ) )
```

The SEIS policy is denoted as:

```
( createConcept SeisAsymmetricKeySizes
    ( and Rfc2527AsymmetricKeySizes
          ( all keyLength ( minimum  1024 ) ) ) )
```

Using the built-in properties of NEOCLASSIC's type `Integer` an ordering scheme is predefined: `Rfc2527AsymmetricKeySizes` defines a class of instances with the attribute `keyLength`, which may take all integers as value; `GocSign1AsymmetricKeySizes` restricts the values to greater equal 512; `GocSign[23]AsymmetricKeySizes` restricts the values to greater equal 1024; `GocSign4AsymmetricKeySizes` restricts the values to greater equal 2048. Because of the properties of `Integer` the application of the subsumption reasoning service results in the following ordering:

– `Rfc2527AsymmetricKeySizes` subsumes `GocSign1AsymmetricKeySizes`
– which in turn subsumes `GocSign[23]AsymmetricKeySizes`
– which in turn subsumes `GocSign4AsymmetricKeySizes`

Given the defined semantics of the taxonomy end users can conclude that certificates issued under security policy `GocSign4AsymmetricKeySizes` can be relied on most.

NEOCLASSIC also automatically determines that

– `GocSign[23]AsymmetricKeySizes` is equivalent to `SeisAsymmetric-KeySizes`

If the only requirement would be to compare numeric values the general subsumption mechanism would not be necessary - PICS [8] does something similar. However, not all properties described in a policy can be represented by numeric values, which can be seen in the next examples.

4.2 Key Pair Generation

In order to support comparison operations it is necessary to define an order relationship amongst newly defined concepts.

Example: a key pair generated in hardware might be more trustworthy than a key pair generated in software.

```
( createConcept Rfc2527Sw     Rfc2527ModuleTypes )
( createConcept Rfc2527HwOrSw Rfc2527Sw )
( createConcept Rfc2527Hw     Rfc2527HwOrSw )
```

Above statements define that

– `Rfc2527Hw`, `Rfc2527HwOrSw`, `Rfc2527Sw` are sub-classes of `Rfc2527ModuleTypes`
– `Rfc2527Sw` subsumes `Rfc2527HwOrSw`
– `Rfc2527HwOrSw` subsumes `Rfc2527Hw`.

This taxonomy is now associated with a security related semantic: module types whose concept descriptions subsume others are less trustworthy than the module types associated with the subsumed concepts.

Using this taxonomy `Rfc2527KeyGeneration` is defined with two attributes: `caKeyGen` and `eeKeyGen` (generation of keys for CA operations and generation of keys for end user operations respectively). Both of which require values that belong to the concept `Rfc2527ModuleTypes`.

```
( createConcept Rfc2527KeyGeneration
    ( and ( all caKeyGen Rfc2527ModuleTypes )
          ( all eeKeyGen Rfc2527ModuleTypes ) ) )
```

The Government of Canada policy can be specified as:

```
( createConcept GocSign2KeyGeneration
    ( and Rfc2527KeyGeneration
          ( all caKeyGen Rfc2527HwOrSw )
          ( all eeKeyGen Rfc2527HwOrSw ) ) )

( createConcept GocSign3KeyGeneration
    ( and Rfc2527KeyGeneration
          ( all caKeyGen Rfc2527Hw     )
          ( all eeKeyGen Rfc2527HwOrSw ) ) )
( createConcept GocSign4KeyGeneration
    ( and Rfc2527KeyGeneration
          ( all caKeyGen Rfc2527Hw     )
          ( all eeKeyGen Rfc2527Hw     ) ) )
```

NEOCLASSIC will determine that `GocSign2KeyGeneration` subsumes `GocSign3KeyGeneration`, which subsumes `GocSign4KeyGeneration`. This can then in turn be interpreted in such a way that certificates associated with policy `GocSign4` are more secure than certificates associated with policy `GocSign2`.

The SEIS policy can be described as:

```
( createConcept SeisKeyGeneration
    ( and Rfc2527KeyGeneration
          ( all caKeyGen Rfc2527HwOrSw )
          ( all eeKeyGen Rfc2527HwOrSw ) ) )
```

NEOCLASSIC will recognize `SeisKeyGeneration` as being equivalent to `GocSign2KeyGeneration`.

5 Summary

This paper shows how description logics can be used to represent certificate policy information. It has been discussed how subsumption can be used in order to compare the quality and trustworthiness of certificates.

Performing the case study of modeling different policy the following observations have been made:

– The definition of a core terminology in form of an ontology is necessary. RFC 2527 actually provides some kind of framework that can be followed to specify such a reference terminology.
– It also became apparent that policies that follow RFC 2527 are difficult to compare because this framework leaves too much room for interpretation and expressing different aspects.
– This shows that users who want to compare the quality of certificates actually do face a major problem, as existing policies are difficult to compare.

Planned future work comprises the investigation on the use of more expressive DL-systems and how a DL-based language can best be syntactically represented.

While performing this work it also became apparent that different domains are using varying semantic data models to represent authorizations, capabilities, rights, etc. These different representations in turn require domain-specific processing algorithms. It seems promising to study how a DL-based system can be used as a unifying scheme for a generic policy specification.

6 References

1. Housley, R., Ford, W., Polk, W., Solo, D.: Internet X.509 Public Key Infrastructure: Certificate and CRL Profile, IETF RFC 2459, 1999.
2. Chokhani, S., Ford, W.: Internet X.509 Public Key Infrastructure: Certificate Policy and Certification Practices Framework, IETF RFC 2527 (1999)
3. Donini, F. M., Lenzerini, M., Nardi, D., Schaerf, A.: Reasoning in Description Logics, CLSI Publications, Principles of Knowledge Representation and Reasoning, (1994) 193-238
4. Borgida, A., Patel-Schneider, P. F.: A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic, Journal of Artificial Intelligence Research 1, (1994) 277-308
5. Lassila, O., Swick, R.: Resource Description Framework (RDF): Model and Syntax Specification, W3C Recommendation, (1999)
6. Treasury Board of Canada, Secretariat: Digital Signature and Confidentiality Certificate Policies for the Government of Canada Public Key Infrastructure", Version 3.02 (1999)
7. Secured Electronic Information in Society: SEIS Certificate Policy SeisS10-1: 1.0, High assurance general ID-certificate with private key protected in an electronic ID-card, Version 1.0 (1998)
8. Krauskopf, T., Miller, J., Resnick, P., Treese, W.: PICS Label Distribution Label Syntax and Communication Protocols, W3C Recommendation (1996)
9. Harmelen, F. van, Patel-Schneider, P., Horrocks, I., "Reference description of the DAML+OIL (March 2001) ontology markup language", http://www.daml.org/2001/03/reference.html, March 2001

A hybrid approach to extend DL-based reasoning with concrete domains

Bo Hu[1] Ernesto Compatangelo[2] Ines Arana[1]

[1] School of Computer and Mathematical Sciences,
The Robert Gordon University, Aberdeen, UK

[2] Department of Computing Science, University of Aberdeen, UK

29th August 2001

Abstract

We propose a new hybrid approach which extends the expressive power of DL languages by incorporating concrete domains. Without modifying the DL inference algorithms, our approach uses the results of other non-DL inferential engines to reason about terminological knowledge. Constraints involving concrete domains are reasoned and replaced with equivalent concept restrictions exclusively based on the expressive power of the DL languages selected as the DL-based inferential engines. Meanwhile, We outline a system architecture that can support such an approach, which involves a homogeneous knowledge representation and hybrid reasoning.

1 Extending existing DL-based system

Description Logics (DLs) are a well-known family of knowledge representation and reasoning formalisms [6]. They are featured by the ability of building up complex knowledge from basic notion of concepts (unary predicates) and roles (binary relations). Various DL-based systems are also available to provide inferences on such complex knowledge. Ever since they were introduced two decades ago, DLs have always been characterised by a reasonable trade-off between expressive power and computational complexity. However, this trade-off has not prevented the development of inferential engines based on expressive concept languages, such as $\mathcal{SHIQ}$ [10].

So far, the major effort in extending the expressive power of DL systems has been put in the enhancement or optimisation of the inference algorithms. Various extensions have been proposed or implemented in the last decade. For instance, concrete domains have been introduced into DLs which are normally used to capture abstract domains [2]. A very expressive language which provides reasoning services on individuals and limited concrete knowledge (concrete data types) has also been introduced [5]. Despite their diversity, virtually all of the above approaches, which are based on the $\mathcal{ALC}$ language [13], extend the original $\mathcal{ALC}$ tableau inference algorithm in different ways [3].

Modifying the inference algorithm normally results in a well-performed new system with new or improved underlying inference capabilities. However, such approach is not very good at handling situations where knowledge engineers want to use the existing DL systems without touching anything "inside", enhancing them with proper extensions in a task-specific way. These situations provide reasonable motivations for the introduction of our approach.

In this paper, we neither introduce a revised DL-based modelling language nor a new inference algorithm, but a formal scheme which can be applied to existing DL systems. In another word, we focus on a top-up system which can be easily implemented and tailored to a particular application. Meanwhile, we intend to use such a system as the workbench to explore the role of hybrid approach in extending the expressive power of DLs.

Since we intend to keep our approach as portable and generic as possible, we deliberately do not use certain features provided by some DL systems. For instance, the CLASSIC Terminological Knowledge Representation and Reasoning System (TKRRS) [4] allows users to query the domain of a particular role while other TKRRS, such as FaCT [9] do not implement such mechanisms. Therefore, we have decided to avoid using them.

2 Representing concrete domains

The abstract characters of DLs make it difficult to naturally model the knowledge on concrete domains such as arithmetic ones while in certain applications, modelling on the concrete characters such as age, size, weight, etc is necessary.

Including concrete domains into DLs is always a very interesting issue. Previous efforts such as $\mathcal{ALC}$ (D) [2] and $\mathcal{ALCRP}$ (D) [8] allow concrete properties to be referred through functional role chains. For instance, if we want to constrain that the service time of an apprentice is no more than 5 years, we can defined "apprentice" as $\mathsf{Apprentice} \doteq \mathsf{Human} \sqcap \exists \mathsf{employer} \sqcap \mathsf{5year}(\mathsf{len\text{-}of\text{-}serv})$, where $\mathsf{len\text{-}of\text{-}sev}$ is the functional role. The predicate $\mathsf{5year}$ is defined using functions written in the host programming languages. One way to implement it can be: $\mathsf{5year}(x) = \ x \geq 5$. The approach of $\mathcal{ALC}$ (D) provides DLs a perfect means to describe the constraints on concrete domains, however, with high computational complexity [11]. Such situation inspired us to contrive a systems with limited increment on the complexity while obtain the similar expressive power, i.e. in our case, the expressive power of the concrete properties.

In this paper, we focus on concrete domains which are formally defined as followings:

A concrete domain $\mathcal{D}$ is a pair $\mathcal{D}$ =($\Delta_{\mathcal{D}}$, $\Phi_{\mathcal{D}}$), where

1. *$\Delta_{\mathcal{D}}$ is a set called domain,*
2. *$\Phi_{\mathcal{D}}$ is a set of predicates;*
3. *Each predicate $P \in \Phi_{\mathcal{D}}$ is associated with an arity n and an n-ary predicate $P^{\mathcal{D}} \subseteq \Delta_{\mathcal{D}}^{n}$.*

For the sack of simplicity, we will restrict the predicates to those arithmetic and comparison ones and the domain $\Delta_{\mathcal{D}}$ to a finite one. Nevertheless, we are quite aware that Boolean operators can be introduced easily to form more complex expressions (e.g. meta-constraints). Furthermore, other concrete domains will be considered in successive research.

In this section, we will give the syntax and semantics of our conceptual modelling language. Since we are not restricted to a particular DL, here and in the following, we take $\mathcal{ALC}$ for the demonstration purpose. Nevertheless, such approach can be applied to virtually any DL systems.

Constructor	**Syntax**	**Semantics** (*Interpretation*)
Top (Universe)	$\top$	$\Delta^{\mathcal{I}}$
Bottom (Nothing)	$\bot$	$\emptyset$
Atomic Concept	A	$\mathsf{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Atomic Role	R	$\mathsf{R}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Conjunction	$\mathsf{C} \sqcap \mathsf{D}$	$\mathsf{C}^{\mathcal{I}} \cap \mathsf{D}^{\mathcal{I}}$
Disjunction	$\mathsf{C} \sqcup \mathsf{D}$	$\mathsf{C}^{\mathcal{I}} \cup \mathsf{D}^{\mathcal{I}}$
Negation	$\neg\, \mathsf{C}$	$\Delta^{\mathcal{I}} \setminus \mathsf{D}^{\mathcal{I}}$
Universal quantification	$\forall \mathsf{R}.\mathsf{C}$	$\{\, c \in \Delta^{\mathcal{I}} \mid \forall\, d \in \Delta^{\mathcal{I}} : \langle\, c, d\,\rangle \in \mathsf{R}^{\mathcal{I}} \rightarrow d \in \mathsf{C}^{\mathcal{I}} \,\}$
Existential quantification	$\exists \mathsf{R}.\mathsf{C}$	$\{\, c \in \Delta^{\mathcal{I}} \mid \exists\, d \in \Delta^{\mathcal{I}} : \langle\, c, d\,\rangle \in \mathsf{R}^{\mathcal{I}} \wedge d \in \mathsf{C}^{\mathcal{I}} \,\}$

Table 1: $\mathcal{ALC}$ syntax and semantics

Let R be a role, H the constrained element, P the defined predicate name. In addition to the syntax of $\mathcal{ALC}$ [13] (Table 1),

1. *$\forall \mathsf{R}.\mathsf{H}$ in a concept, where H acts as the bridge between the abstract and the concrete domains;*
2. *P(H_1, .. , H_n) specifies the constraint on H_1, .. , H_n; and CE(H_1, .. , H_n) is the constraint expression which collects all the constraints on H_1, .. , H_n or a subset of them;*

H_i ($i \leq n$) which are referred to as Hybrid Concepts are defined by giving each of them a unique name and mapping it to a subset of $\Delta_{\mathcal{D}}$.

In order to associate abstract and concrete domains, we introduce an assignment function $\lambda: \ \Delta^{\mathcal{I}} \rightarrow \Delta_{\mathcal{D}}$, i.e. $\lambda(\mathrm{H}^{\mathcal{I}}) \subseteq \Delta_{\mathcal{D}}$ which maps every *Hybrid Concept* to a subset of $\Delta_{\mathcal{D}}$. Thus, the semantics is given as

Let $\cdot^{\mathcal{I}}$ be the interpretation function, $\lambda(\cdot)$ the assignment, and $\Delta^{\mathcal{I}}$ a non-empty domain which is disjoint from $\Delta_{\mathcal{D}}$. In addition to the semantics of $\mathcal{ALC}$ [13] (Table 1),

1. $(\forall \mathsf{R}.\mathsf{H})^{\mathcal{I}} \ = \ \{x \in \Delta^{\mathcal{I}} \mid \forall y.\langle x, \ y\rangle \in \mathsf{R}^{\mathcal{I}} \ \rightarrow \ y \in \mathsf{H}^{\mathcal{I}}\}$;

2. *$P(\mathsf{H}_1, \ldots, \mathsf{H}_n)$ is satisfied iff*

$$\forall s_1 \in \lambda(H_1^{\mathcal{I}}), ..., \forall s_n \in \lambda(H_n^{\mathcal{I}}).P(s_1 \ ... \ s_n);$$

3. *$CE(\mathsf{H}_1, \ldots, \mathsf{H}_n)$ is satisfied iff*

$$\begin{aligned}&\forall s_1 \in \lambda(H_1^{\mathcal{I}}), ..., \forall s_n \in \lambda(H_n^{\mathcal{I}}).\\ &\qquad (\forall P \in CE(\mathsf{H}_1, \ldots, \mathsf{H}_n).P(s_1 \ ... \ s_n)).\end{aligned}$$

By introducing the assignment function between abstract and concrete domains, we are able to embed concrete knowledge into so-called "*Hybrid Concepts*". Such embedding has non-trivial consequences, as referring to concrete knowledge in concept definitions is no longer required. More specifically, those inference algorithms which are designed to tackle with concrete domains become unnecessary, in that "wrapper concepts" (please refer to Section 3) which are the "incarnations" of *Hybrid Concepts* in DLs are introduced as atomic concepts and treated exactly the same way as normal concepts. In stead of reasoning with *Hybrid Concepts*, system replaces each of them with the wrapper concept. Meanwhile, the responsibility of reasoning with the concrete knowledge represented by *Hybrid Concepts* is delivered to the reasoning systems other than DL ones. For instance, in the above example the apprentice's length-of-service can be modelled as *Hybrid Concepts* represented using universal quantification in the hybrid concept definitions.

Thus, we have apprentices defined as

$$\mathsf{Apprentice} \doteq \mathsf{Human} \sqcap \exists \mathsf{employer} \sqcap \forall \mathsf{len\text{-}of\text{-}serv}.\mathsf{Year}_{\mathsf{App}},$$

where $\mathsf{Year}_{\mathsf{App}}$ is *Hybrid Concept.* Moreover, we can specify slightly intricate facts by putting constraints on $\mathsf{Year}_{\mathsf{App}}$. For instance, a specialist is defined as

$$\mathsf{specialist} \doteq \mathsf{Human} \sqcap \exists \mathsf{employer} \sqcap \forall \mathsf{len\text{-}of\text{-}serv}.\mathsf{Year}_{\mathsf{Spl}}.$$

Note that: at current stage, we do not restrict the *Hybrid Concepts* to be successors of functional roles (features), but we require that the role value restriction ($\forall$) is used instead of the role existence restriction ($\exists$). Ignoring the latter is motivated by pragmatic considerations. To simulate partial constrained role values, array-type variables may be required. However, partial orders among arrays are not straightforward.

The specifications and restrictions on the *Hybrid Concepts* are given separately as the conjunction of non-DL expressions:

$$\mathsf{Year}_{\mathsf{App}} \leq 5 \wedge \mathsf{Year}_{\mathsf{Spl}} \geq \mathsf{Year}_{\mathsf{App}} + 10.$$

In such an approach, more complex constraints than min and max can be specified. However, such constraints have to be global restrictions which hold universally. In another word, the constraints have to hold on a concept as a group rather than considering each of its instance individually. For instance, as long as one restricts the diameter of a certain type of cylinder, its cross-sectional area is constrained. Also, if we restrict the age of pupils and the average age difference between them and their parents, we constrain the age of the latter to a certain range. Of course, such difference can exist in a much more complex way than "20 years older". Specifying the universal constraints separately, we are granted the freedom to add, delete, change and satisfy them in a batch job.

Apparently, if we want to avoid the complex interaction between DL and non-DL expressions, no available DL system can properly handle such expressions. However, we can hide the actual inferences on the concrete domains from DL systems. More specifically, we allow conceptual hierarchies to be simulated by partial orders which are more general than subsumption relationships between concepts. In our case, with the help of formal numeric systems, we can create hierarchies of *Hybrid Concepts* (more detailed examples can be found in the next section). As a result, DL system is informed of such hierarchies through the subsumption relationship between the wrapper concepts. It then classifies the concepts (such as $\mathsf{Apprentice}$, $\mathsf{Specialist}$) which are built using the wrapper concepts.

Here and in the following, we will use Constraint Programming Languages (CPLs, also referred to as Constraint Solvers, CSs for short, in certain cases) for illustration purposes. Nevertheless, we notice that the actual applications are not restricted to such reasoning engines.

3 Tackling with the "wrapper concepts"

For the sake of simplicity and integrity, we use hybrid concept definitions to capture both abstract and concrete knowledge. However, as pointed in the previous section, the latter has to be wrapped in order to be properly manipulated. Concepts which contains wrapped concrete knowledge are referred to as normalised concepts. They are defined in a broader sense as followings:

> *A normalised heterogeneous concept definition (**normalised concept**, for short) is the result of a normalisation process that for every occurrence of Hybrid Concepts an atomic concept will be introduced with which the former will be replaced. Such atomic concepts are referred to as "**w**rapper concepts".*

Hybrid concept definitions contained in a *Hybrid Knowledge Base* (HKB) are first analysed and processed by a *parser* which normalises the definitions and generates three sets of statements. These three sets are defined as followings:

- a set of DL-oriented statements which do not exceed the expressive power of the external DL system (warnings will be generated otherwise),
- a set of non-DL statements which express all the concrete knowledge,
- a set of *Hybrid Concepts* which connect DL and non-DL statements.

The hybrid characteristic of our approach is evident in the "polymorphism" of *Hybrid Concepts*. More specifically, *Hybrid Concepts* are represented by wrapper concepts in DL inferential engines while act as legal objects in non-DL reasoning systems, (e.g. constrained variables in CPLs).

For instance, let us suppose that in certain state X married persons are required to be 22-year-old or older. We define concept Married as those "legally married people", Golden-Couples as "people who have already celebrated their golden (50th) anniversary", and Senior-Citizen as "people who are 70 and older". Together with other necessary concepts, we have

$$\begin{aligned} \mathsf{Married} &\doteq \mathsf{Human} \sqcap (\exists \mathsf{has\text{-}spouse.Human}) \sqcap (\forall \mathsf{age.Age_{Mar}}) \\ \mathsf{Golden\text{-}Couples} &\doteq \mathsf{Human} \sqcap (\exists \mathsf{has\text{-}spouse.Human}) \sqcap (\forall \mathsf{married\text{-}year.Year50}) \\ &\quad \sqcap (\forall \mathsf{age.Age_{Gol}}) \\ \mathsf{Senior\text{-}Citizen} &\doteq \mathsf{Human} \sqcap (\forall \mathsf{age.Age_{Sen}}) \end{aligned} \tag{1}$$

Meanwhile, the restrictions on all the *Hybrid Concepts* are given as

$$\begin{aligned} \mathsf{Age_{Mar}} &\geq 22 \\ \mathsf{Year50} &= 50 \\ \mathsf{Age_{Sen}} &\geq 70 \end{aligned} \tag{2}$$

Because the actual constraint exists between two sets of values, rather than write an inequality, we use the relation between sets instead:

$$\mathsf{Age}_{\mathsf{Gol}} - \mathsf{Year50} \subseteq \mathsf{Age}_{\mathsf{Mar}} \tag{3}$$

Now, by normalising the knowledge base we split the above definitions and restrictions into three parts. First, we replace all the *Hybrid Concepts* with "wrapper concepts" and adding new atomic concepts, $\mathsf{Age}_{\mathsf{Mar}}$, $\mathsf{Age}_{\mathsf{Gol}}$, $\mathsf{Age}_{\mathsf{Sen}}$, and $\mathsf{Year50}$ into the DL part. Second, all "ages" acting as constrained variables are stored in the non-DL part together with their default domain [0..100][1] and the constraints defined in (2). Assuming that $\mathsf{domain}()$ is the assignment function, we can specify for each *Hybrid Concepts*:

$$\begin{array}{ll} \text{non-DL part:} & \\ & \mathsf{domain}(\mathsf{Age}_{\mathsf{Mar}}) = [0..100] \\ & \mathsf{domain}(\mathsf{Age}_{\mathsf{Gol}}) = [0..100] \\ & \mathsf{domain}(\mathsf{Age}_{\mathsf{Sen}}) = [0..100] \\ & \ldots \end{array} \tag{4}$$

Hybrid Concepts are also stored separately in the so-called "Link Pool", (see Figure 2 for the "Link Pool").

The above statements are translated into the underlying modelling languages of the external inferential engines. Such translations are carried out so as to keep our approach portable and implementation-independent. Subsequently, translated statements are loaded into external DL and CPL inferential engines. According to the results provided by these different engines, a reasoning coordinator (see Figure 3) creates hierarchical structures of *Hybrid Concepts*, which are then introduced into DL definitions through the "wrapper concepts".

In our example, after loading the non-DL part (2)+(3)+(4) into an external constraint solver, we obtain the reduced domains (e.g. $\mathsf{domain}(\mathsf{Age}_{\mathsf{Gol}}) = [72..100]$). Using the results from both DL and non-DL inferential engines, we create a new partial order among "ages" e.g. $\mathsf{domain}(\mathsf{Age}_{\mathsf{Gol}}) \subseteq \mathsf{domain}(\mathsf{Age}_{\mathsf{Sen}})$. Therefore, the corresponding subsumption relationships can be specified between wrapper concepts (e.g. $\mathsf{Age}_{\mathsf{Gol}} \sqsubseteq \mathsf{Age}_{\mathsf{Sen}}$). Sending such information back to join the original DL definitions contained in the DL part in (1), we can conclude that, among other conclusions, in the state X, people who have already celebrated their 50th anniversary are all senior citizens, i.e.

$$\mathsf{Golden\text{-}Couples} \sqsubseteq \mathsf{Senior\text{-}Citizen}.$$

[1]We assume that human can not live out 100.

Thinking globally, such conclusion is evident in the sense that although people get married and celebrate their Golden Anniversary (if there is one) in different ages, the ages of golden couples will always fall into the given range.

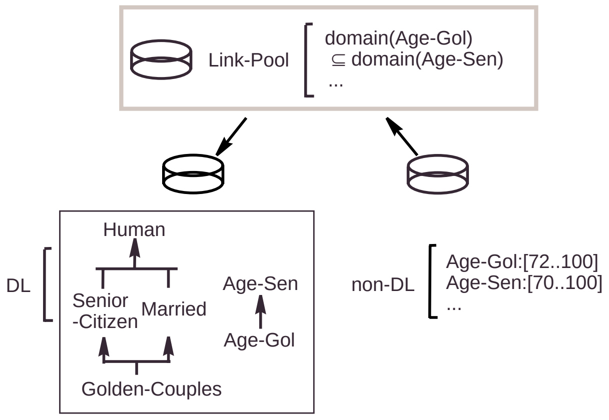


Figure 1: Reasoning with of "ages"

An obvious result of the above reasoning process is that, by satisfying the constraints, we explicitly express some knowledge which otherwise, remains implicit. In our case, by satisfying the constraints, our system can answer the query that "Is Golden-Couples a Senior-Citizen". Although the above "ages" example is a relatively simple one, more complex hybrid knowledge can be represented and processed using the same approach.

4 System Architecture

Since the heterogeneous knowledge used in our approach is evenly split into two separate "homogeneous" components, we use a hybrid architecture to provide the overall inference services. In our system (see Figure 2), the external DL inferential engine and the non-DL one (e.g. a constraint solver in our case) are used in a "peer to peer" way, i.e. neither of them acts as a client or a server with regard to the other one.

However, several points need to be remarked. First, to ensure the portability of our system, we introduce a decoupling between the DL and non-DL representations on one side and the actual inferential engines on the other. More specifically, the transformation process from a hybrid description to the one accepted by the selected DL or non-DL system is divided in two stages. During a first stage, the hybrid language is split into its two homogeneous DL and non-DL components. During a second stage, the two resulting components is actually expressed in an implementation-independent "intermediate" language.

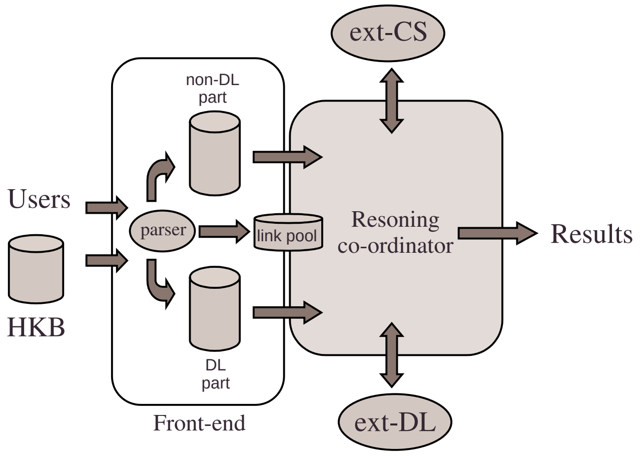


Figure 2: Hybrid architecture

Each of the two intermediate descriptions are subsequently transformed into the ones accepted by the adopted inferential engines. In this way, we decouple our system from the reasoners, thus allowing their modular replacement if necessary. Second, a *link pool* is created to store related data about each newly created "wrapper concept", such as name, position, and so on. This information can be used when "wrapper concepts" must be reclassified according to the results provided by the external inferential engines. Third, various systems can be used as external inferential engines. In this paper, we only analyse the situation where CPLs are selected to reason with numeric restrictions (such as $\mathsf{Age}_{\mathsf{Gol}} - \mathsf{Year50} \subseteq \mathsf{Age}_{\mathsf{Mar}}$ in our example) for the demonstration purposes.

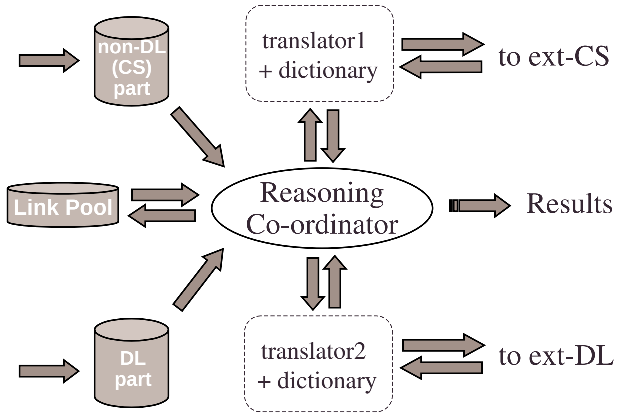


Figure 3: Reasoning Co-ordinator

5 Conclusions

We give an overview of a novel approach to extend the expressive power of existing DLs. This approach is based on a hybrid reasoning process. Thus, we also propose an architecture to support it.

The advantages of a hybrid system are system portability, extensibility, and robustness. Although similar approaches have been already proposed in the past (e.g. TexLog [1]), our approach benefits from the heterogeneous characteristics of *Hybrid Concepts*, which have a consistent semantics on both DL and non-DL (e.g. CPLs) sides. Such a bridging facility also differentiates our approach from other apparently similar ones.

At current stage, we haven't examined the complexity issue thoroughly. Nevertheless, there is great potential to optimized our system by tailoring the DL and CPLs inferential engines to particular applications. Such character is enhanced by selecting the external inferential engine with the right expressive power. In practice, the overall performance of our system can be considered separately. On the one hand, since we do not introduce any new constructors– only new concepts, we expect that the complexity of the DL inference will be in the same class as the original system. Moreover, we avoid the complex interaction between abstract and concrete domains by introducing the latter through "wrapper concepts". On the other hand, the Finite Constraint Satisfaction Problems (FCSPs) are NP-complete as a general class [12]. Pragmatic results shows that the performance varies from system to system [7]. However, selecting the suitable CPLs is not the concern of this paper.

Because of the adopted hybrid approach, the overall performance of our system could be influenced by the various translation and interfacing processes. This might be particularly evident if several different protocols must be used to link each of them to the other as well as to the controller. This drawback may not be a serious problem in all those cases which are not time-critical and in which reasoning with heterogeneous knowledge is necessary. However, further analysis on this issue is necessary.

References

[1] Andreas Abecker and Holger Wache. A layer architecture for the integration of rules, inheritance, and constraints. In *ICLP Workshop: Integration of Declarative Paradigms*, pages 12–22, 1994.

[2] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In J. Mylopoulos and R. Reiter, editors, *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence IJCAI-91*, pages 452–457. Morgan Kaufmann Publ. Inc., San Mateo, CA, 1991.

[3] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 2001. To appear.

[4] Ronald J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, and A. Borgida. "Reducing" CLASSIC to practice: Knowledge representation theory meets reality. *Artificial Intelligence 114*, pages 203–237, 1999.

[5] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks. Enabling knowledge representation on the web by extending RDF schema. In *Proceedings of the tenth World Wide Web conference WWWW'10*, 2001. To appear.

[6] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Reasoning in description logics. In Gerhard Brewka, editor, *Foundation of Knowledge Representation*, pages 191–236. CSLI-Publications, 1996.

[7] A. Fernández and P. M. Hill. A comparative study of eight constraint programming languages over the Boolean and finite domains. *Journal of Constraints*, 5:275–301, 2000.

[8] Volker Haarslev, Carsten Lutz, and Ralf Möller. A description logic with concrete domains and role-forming predicates. *Journal of Logic and Computation*, 9(3), 1999.

[9] I. Horrocks. FaCT and iFaCT. In *Proc. of the Int. Workshop on Description Logics (DL'99)*, pages 133–135, 1999.

[10] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.

[11] C. Lutz. NExpTime-complete description logics with concrete domains. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artifical Intelligence, pages 45–60, Siena, Italy, 2001. Springer Verlag.

[12] Alan K. Mackworth and Eugene C. Freuder. The complexity of constraint satisfaction revisited. *Artificial Intelligence*, 59(1–2):57–62, 1993.

[13] M. Schmidt-Schauss and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

POSITION PAPER
Some Requirements for Practical Modeling in Dialogue Systems

Michael Knorr, Bernd Ludwig, Günther Görz
Computer Science Institute and FORWISS, Erlangen, Germany
Email: goerz@informatik.uni-erlangen.de

Abstract

The goal of this paper is to stimulate a discussion between description logic system developers and users of description logic systems with an emphasis on applications in the field of dialogue systems. It consists of two parts: The first part is a report on experiences in temporal representation and calendrical reasoning with description logics from which we derive requirements on decription logics as a useful modelling tool.

The second part which is more speculative in nature complements these considerations with further modelling tasks and corresponding expressive means modelling languages should meet to deal with them in a satisfactory way.

1 Experiences from using Description Logic for Time Representation

1.1 Introduction

Our research group has been involved in the development of speech dialogue systems for many years. These include EVAR [8], a system for train information, and EMBASSI[1] an application for interacting with home entertainment systems. A common feature of these and many other domains is that the exchange of temporal information is an essential part of the dialogue, be it the departure time of a train or the start time of a TV programme. Therefore we decided to investigate the syntax and semantics of German temporal expressions independently of a specific domain.

[1]See the contribution by Bücher et al. in this volume

Nearly all working speech dialogue systems, including the aforementioned EVAR, model dialogues with some kind of finite state machine. An evaluation of dialogues from the EVAR system showed, that such systems lack the flexibility to deal with "meta-dialogues", like corrections or changes in the dialogue goal. As a consequence we decided that the understanding of utterances should rely on well-defined inference procedures to reason about the meaning of utterances and the dynamic evolution of the dialogue structure. In order to obtain the semantics of natural language utterances one should rely on a domain model, defining the actions and objects salient to the application. We decided to use Description Logics (DL) for the construction of the domain model[18].

This leads to the need to represent the semantics of temporal expressions in Description Logics. When we started this work in 1999 we investigated the suitability of four DL-systems, namely CICLOP [17], CLASSIC [19], FaCT [13], and RACE [10] for the representation of temporal information. At this time only CLASSIC fulfilled our requirements. In the following sections we describe the representation of temporal information in our application and discuss what requirements for DL systems arise from it. Implementors of DL-systems might use them to consider what features their systems need to be useful for this domain. Users wishing to employ DL-systems for tasks similar to ours, might consider how far similar requirements apply to their own applications.

1.2 Representing Temporal Information

In this section we give a short description of our semantic model for temporal expressions. The words in our lexicon correspond to concepts in the T-Box of the DL system. When an utterance is parsed an individual is created in the A-Box of the DL system for every word encountered in the expression. The application of a grammar rule leads to either

- the creation of a new individiual, representing the whole phrase, with the individuals representing its parts used as role fillers.
- the selection of a specific individual and its connection to the other individuals in the expression by roles.
- or the mergence of the individuals representing the parts of the expression.

In many cases this triggers (forward-chaining) rules to move information or perform calculations whose results are added to the appropriate individuals.

The top concepts of our concept hierarchy are `TimePoint`, `TimeInterval`, `Duration` and `TimeUnit`. Most temporal expressions describe a point in time. They are represented by merging individuals of `Clocktime` (defining minute and hour), `PartOfDay`, `Date` (defining day, month and year) and `ISO-Date` (defining day of week and week). These concepts are therefore subsumed by `TimePoint`. `Duration` represents the length of an interval and the distance between two time points; intervals are represented by `TimeInterval`. The individuals representing time units, such as minute, hour, day, are of concept `TimeUnit`. Details of the representation can be found in [14].

1.3 Examples of Temporal Expressions

We now show in detail how the representation for a temporal expression is constructed. As example we take: *am siebzehnten dritten zweitausend um dreiviertel vier nachmittags* (on the 17 March 2000 at quarter to four in the afternoon). This expressions consists of three chunks, that could appear in any order. The semantics of the chunks can also be constructed and merged in arbitrary order.

The construction of the semantic representation starts when the word *siebzehnten* is parsed. An individual of concept `Date` is constructed and the attribute `has-day` is filled with 17. In the next steps the attributes `has-month` and `has-year` are set to 3 and 2000. Now a test function (cf. [5]) checks that 17 March 2000 is indeed a valid date in the Gregorian Calendar. Additionally a rule is triggered and calculates the day of week and the ISO week number. If later a day of week would be added which is different from the calculated one, the system would detect that the expression is inconsistent.

The individual now contains the following information:

```
(AND TimePoint (FILLS has-year 2000)
               (FILLS has-month 3)
               (FILLS has-week 11)
               (FILLS has-dow 5)
               (FILLS has-day 17))
```

The representation of the clocktime phrase starts with the construction of the individuals for *dreiviertel* and *vier*. *Dreiviertel* is represented as `SpecialMinutesFrom` with `has-timedistance` 15 Minutes and `has-direction` -1. *Vier* is mapped onto `TraditionalClocktime` with `has-traditional-hour` 4 and `has-minute` 0. A rule then derives that `has-hour` is `(ONE-OF 4 16)`. To represent the combination of *dreiviertel* and *vier*, the individual for the latter is filled in the `has-basetime`-attribute of the former. This triggers more information processing rules. The resulting representation is:

```
(AND SpecialMinutesFrom (ALL has-hour (ONE-OF 3 15))
                        (FILLS has-traditional-hour 3)
                        (FILLS has-minute 45)
                        (FILLS has-timedistance 15)
                        (FILLS has-direction -1)
                        (FILLS has-basetime OtherIndividual)
                        (SAME-AS (has-basevalue)
                           (has-basetime has-traditional-hour)))
```

With `OtherIndividual`:

```
(AND TwentyfourClocktime TraditionalClocktime (ALL has-hour (ONE-OF 4 16))
                        (FILLS has-traditional-hour 4)
                        (FILLS has-minute 0))
```

The adverb *nachmittags* (in the afternoon) leads to the creation of an individual of concept `Afternoon`. This places the constraint `(ALL has-hour (MIN 12)(MAX 18))` on the `has-hour`-role.

When the individual chunks are processed the individuals representing them are merged. The system can now infer that the hour is 15. The individual representing the complete expression is (only an extract is displayed):

```
(AND SpecialMinutesFrom (FILLS has-year 2000)
                        (FILLS has-month 3)
                        (FILLS has-week 11)
                        (FILLS has-day 17)
                        (FILLS has-dow 5)
                        (FILLS has-hour 15)
                        (FILLS has-minute 45)
                        (FILLS has-traditional-hour 3)
                        (FILLS has-basetime OtherIndividual))
```

This is still a fairly simple expression in terms of processing. Some very complex expressions look rather peaceful on the surface, consider *drei Tage vor Christi Himmelfahrt* (three days before Ascension Day). First we need to know the date of Ascension Day in a given year, than we need an algorithm that subtracts three days from it, taking care of month boundaries. To get the date of the holiday we either have to access an external calendar or add 39 days to the date of Easter. There are a number of algorithms to calculate the date of Easter, but they are all mathematically complex. We solved the problem by using an external library for calendrical calculations[7].

1.4 Requirements for a Description Logic for Time Representation

From our experience there are three main requirements for a DL-system to be used for time representation. First it needs to be able to deal with numbers. All information neccessary to represent time can be given using natural numbers.

For some expressions it would be useful to be able to use fractions (think of *quarter to* for example). In CLASSIC it is possible, to integrate objects of the host language, in this case LISP, in the Description Logic. Such host concepts can for example be strings or numbers. We used a number of host concepts based on LISP numbers.

Second we do not only want to represent natural numbers but also restrict their range and use them in simple calculations. Individuals of the concept `hour`, for instance, can be values between 0 and 24. It does not matter, from the applicational point of view, if the range check is applied on roles or concepts. An example for a simple calculation occurs, when we encounter *dreiviertel vier*. Here we have to set the role `has-hour` to 3, decrementing 4 by 1.

Third we either need external function calls or we have to be able to do more complex calculations within the DL-system. This starts with rather simple examples like testing if a date is valid and ends with very complex calculations for example to find the date of Easter in a given year. As mentioned above, test concepts as they have been introduced in CLASSIC, provide an appropriate extensible interface for reasoning with individuals. Intensional reasoning however requires an extension of subsumption for test concepts.

In order to perform reasoning with time intervals, there are two possibilities in principle, for both of which we are not aware of an operational implementation: Temporal Description Logics like $\mathcal{TL} - \mathcal{ALCF}$ (cf. e.g. [3]) or a temporal concrete domain based on Allen's interval algebra [2] as proposed e.g. by Kullmann [15].

A description logic allowing the use of concrete domains is $\mathcal{ALC}(\mathcal{D})$[4]. A well known problem of this logic is that it can become undecidable when transitive roles and role hierarchies are allowed (cf. e.g. [12]). However none of these are necessary in our application. According to [11] the RACER-system will soon offer $\mathcal{ALCQHI}_{R+}(\mathcal{D})^{-}$. A constraint of this logic is, that it does not allow to use feature chains. We have not investigated yet if we can dispense with them in all cases; probably not for simulating feature unification (sse below).

With respect to the use of concrete domains, an analogous situation exists with spatial reasoning, for which there is also a need in a variety of applications. Considerable research work has been done in this field, cf. [9], to quote at least one prominent example. We had the opportunity to try out an experimental implementation done by the Hamburg group, based on CLASSIC. In a project in the field of historical cartography, modelling the Behaim Globe of 1492, we implemented an extension to CLASSIC, based on test concepts, for topological and directional reasoning [6]. However, a spatial reasoning service in a more expressive DL system like RACE is an important desideratum.

2 Modelling Tasks and Further Requirements for Expressive Means of Modelling Languages

2.1 Access and Debugging

A Description Logic system that is used in real applications should also fulfill two further requirements. First, the knowledge assembled in the A-Box of the DL system should be easily accessible from the outside. We might, for example, wish to use the temporal information we collected in the system in a database query. For this we need to access the role-fillers of the individual containing the information. In CLASSIC this can be accomplished easily with the function `(cl-fillers @i{Individualname} 'Rolename)`.

Second, a system should contain functions that allow easy debugging of the knowledge base. In a complex knowledge base it is not always easy to see where errors in the inference process arise. It is therefore useful to have functions that print traces of the inference process and to be able to access the full information collected for an A-Box individual at intermediate states of its assembly. In CLASSIC this is facilated by the function `(cl-ind-expr @i{Individualname})`.

2.2 Simulating Feature Structure Unification

In parsing with unification grammars constraints are expressed as path equations. Instead of representing feature structures in a separate formalism, they can as well be expressed in DL. As a small experiment using CLASSIC showed, the unification of feature structures can then be achieved by means of the **same-as** construct. The experiment however was too small for a comparison of performance times.

In the context of a complex computational linguistics application[2], of course one might debate whether the use of a uniform representation formalism ranging over several processing levels is preferable as opposed to level-specific streamlined representation formalisms. In the latter case, processing might be more efficient, but at the cost of translating the resulting representations between levels. In our particular case, on the semantic, application and discourse pragmatic levels, DL is used anyway. So, it makes sense to consider whether the genuine feature structure representation on the syntactic level could be replaced by a DL representation, in particular, because syntactic analysis and semantic construction with linguistic chunks are closely interlocked in an incremental fashion.

[2]cf. our contribution on a dialogue system application in this volume

2.3 A-Box Reasoning

Resolution of ambiguities, in particular on the semantic and pragmatic levels as well as the completion of partial information – e.g. by means of clarification subdialogues – is a central task in dialogue systems. The linguistic analysis module of the dialogue system constructs semantic representations in the form of Discourse Representation structures (DRSs) which are then being refined to application specific DRSs using a formal domain ontology. These instantiated DRSs are the A-Box items on which various reasoning steps must be applied to solve the mentioned resolution and completion tasks. This means that beyond the standard DL reasoning services application domain specific rules are to be applied. Whereas some cases may be covered by forward-chaining rules as e.g. provided by CLASSIC, the general case is to complete subgoals generated by general dialogue goals. In other words, we need additional expressive means for dialogue (step) planning. From an abstract point of view, a language like Reiter's GOLOG language, which has been implemented on top of PROLOG, would provide sufficient means for this purpose. This in turn would lead to the requirement of an A-Box reasoning facility on the basis of Horn rules (cf. CARIN [16]).

As a framework for processing partial information, we found out that FIL [1] meets all our requirements. We started with the implementation of a prover for a Horn clause subset of FIL in Prolog technology, which has later been replaced by a tableau-based reasoner, operating as a separate module. Whether such a service can be integrated as a kind of augmented A-Box reasoner with a DL system is an open question we pose to the DL community.

References

[1] N. Abdallah, *The Logic of Partial Information*. New York: Springer, 1995

[2] J.F. Allen, *Maintaining Knowledge about Temporal Reference*. CACM Vol. 26 Nr. 11, 832–843, 1983

[3] A. Artale, E. Franconi, *Temporal Description Logics*, 1998. To appear in: Handbook of Time and Temporal Reasoning in Artificial Intelligence.

[4] F. Baader, P. Hanschke, *A Scheme for Integrating Concrete Domains into Concept Languages*. Proceedings of the 12th International Joint Conference on Artificial Intelligence, IJCAI-91, 452–457, Sydney, 1991.

[5] A. Borgida, C.L. Isbell, D.L. McGuinness, *Reasoning with Black Boxes: Handling Test Concepts in CLASSIC*. Proceedings of the Description Logics Workshop, 1996

[6] D.M. Deang, *Geometrical and Logical Modelling of Cartographic Objects*. Master Thesis in Computational Engineering, Computer Science Institute, University of Erlangen-Nuremberg, 2000

[7] N. Dershowitz, E.M. Reingold, *Calendrical Calculations*. Cambridge University Press, 1997

[8] F. Gallwitz et al., *The Erlangen Spoken Dialogue System EVAR: A State-of-the-Art Information Retrieval System*. In: *Proceedings of 1998 International Symposium on Spoken Dialogue - ISSD'98*. Sydney, 1998, 19–26

[9] V. Haarslev, R. Möller, *Spatioterminological Resoning: Subsumption Based on Geometrical Inferences*. Proceedings of the Description Logics Workshop, 1997, 74–78

[10] V. Haarslev, R. Möller, *RACE User's Guide and Reference Manual Version 1.1*. Memo-HH-M-289/99, Computer Science Dept., University of Hamburg, 1999

[11] V. Haarslev, R. Möller, *Description of the RACER System and its Applications*. To appear in: Proceedings International Workshop on Description Logics (DL-2001), Stanford, 2001.

[12] V. Haarslev, R. Möller, M. Wessel, *The Description Logic ALCNHR+ Extended with Concrete Domains: A Practically Motivated Approach*. International Joint Conference on Automated Reasoning, IJCAR'2001, Siena. Berlin: Springer, 2001

[13] I. Horrocks, *FaCT Reference Manual Version 1.6*. Computer Science Dept., University of Manchester, 1998

[14] M. Knorr, *Sprachliche und logische Zeitrepräsentation für Dialogsysteme*. Studienarbeit, Universität Erlangen-Nürnberg, 2000

[15] M. Kullmann, *Applying Description Logics to Decision Support*. Ph.D. thesis, LIIA-ENAIS, Strasbourg, 2001

[16] A.Y. Levy, M.-C. Rousset, *Combining Horn rules and description logics in CARIN*. Artificial Intelligence Journal, Vol. 104, 1998, 165–209

[17] LIIA-ENSAIS, *Usermanual for CICLOP Version 1.3b*. Strasbourg, 1999

[18] B. Ludwig, G. Görz, H. Niemann, *An Inference-Based Approach to the Interpretation of Discourse*. First Workshop on Inference in Computational Semantics (ICoS-1), Amsterdam. Jl. of Language and Computation, Vol. 1, No. 2 (2000), 241–258

[19] L.A. Resnick et al., *CLASSIC Description and Reference Manual for the COMMON LISP Implementation: Version 2.3*. AI Principles Research Department, AT&T Bell Laboratories, 1995

Interpolation based Assertion Mining

Stefan Schlobach*
Department of Computer Science, King's College London, Strand
London WC2R 2LS, UK
Email: schlobac@dcs.kcl.ac.uk

Abstract

In this paper we describe a new method for learning terminological knowledge from assertions in description logic based knowledge bases. We present *assertion mining* as the search for *generalised decision concepts (GDC)* which can be used to conceptually define classes of ABox individuals. We show that GDCs can be constructed from ABox interpolants and present tableau based algorithms.

In the recent past learning and knowledge discovery using description logics (DL) have attracted much research. *Least common subsumer (lcs)*–learning has been the most popular approach [3, 1]. lcs–learner construct minimal descriptions for the common properties of a set of positively classified objects in the ABox which do not instantiate any negative example. Unfortunately it is not always possible to construct such a minimal description, it can easily be seen that the *most specific concept* does not exist in DLs with existential quantification and many authors discuss approximations of the knowledge about individuals in knowledge base of different expressivity [8].

In this paper we present *assertion mining* for $\mathcal{ALC}$ knowledge bases as an alternative learning method which shifts the focus from learning of common properties of a set of positive examples towards the construction of discerning concepts which separate positive from negative examples.An advantage of such an approach is that these discerning concepts exist for $\mathcal{ALC}$ even if the most specific concept does not. As in lcs–learning we consider a supervised learning scenario: the ABox of a DL knowledge base contains a number of objects which are classified into positive and negative examples. For each of these classes (which we call decisions) we try to find terminological definitions which can be used to classify new data using instance checking. A concept which preserves the

*This research has been supported by EPSRC grant GR/L91818.

classification properties of a decision will be called *generalised decision concept (GDC)*.

GDCs can be constructed from ABox interpolants for the positive and negative examples. An ABox interpolant is a concept which discerns two elements a and b in a knowledge base Σ, i.e. a concept for which a is an instance of, and for the negation of which b is. We define tableau based algorithms to calculate ABox interpolants for positive and negative examples. It can be shown that if such concepts exist in $\mathcal{ALC}$, they can effectively be calculated even if the most specific concepts do not exist.

Consider the following scenario: A hybrid knowledge base Σ^{Arr} contains information about patients suffering from cardiac arrhythmia. A patient record might include general assertions about gender, age or habits and family information, e.g. $\textsc{pat}_1$: $\textsc{Male} \sqcap \neg\textsc{Smoker} \sqcap \neg\textsc{Old} \sqcap \exists\textsc{hasrelative}.\textsc{Arrhythmia}$, but also technical details about ECG measures such as $(\textsc{pat}_1,\textsc{pw})$:$\textsc{has\_pwave}$; $\textsc{pw}$:$\neg$OK; $\textsc{pat}_2$:$\forall\textsc{has\_pwave}$.OK, and additional knowledge describing the patients' conditions as diagnosed by some medical experts: $\textsc{pat}_1$:$\textsc{Tachycardic}$; $\textsc{pat}_2$:$\textsc{Healthy}$. Assertion mining is the search for terminological axioms which formally define a medical condition and possible diagnostic criteria for these conditions.

We assume familiarity with description logic representation and reasoning and will only briefly introduce $\mathcal{ALC}$. For concepts, which are interpreted as subsets of a universe U, we will usually use the letters C and D, furthermore R for roles, which are interpreted as binary relations over U. The constructors conjunction $C \sqcap D$, disjunction $C \sqcup D$, negation $\neg C$ and existential $(\exists R.C)$ and universal $(\forall R.C)$ quantification have the usual set theoretic interpretation. An ABox $\mathcal{A}$ is a finite collection of role $((a, b) : R)$ or concept $(a : C)$ assertions, where a and b denote individual objects of U. A TBox $\mathcal{T}$ is a set of axioms $C \doteq D$ or $C \dot{\sqsubseteq} D$. A knowledge base Σ is now defined as the pair $(\mathcal{T}, \mathcal{A})$. The reasoning services which we use in this paper are subsumption $\Sigma \models C \sqsubseteq D$ and instance checking $a \in_\Sigma C$ which semantically correspond to the subset and element relation. In $\mathcal{ALC}$, both subsumption and instance checking can be reduced to ABox inconsistency $\Sigma \models \mathcal{A} = \bot$. Example 1 describes an $\mathcal{ALC}$ knowledge base Σ^{arr} with some standard reasoning.

1 Assertion Mining

We will briefly define the main terminology for assertion mining, but refer to a detailed discussion in [12]. We assume that the ABox of a knowledge base[1] consists of a significant amount of possibly noisy information about individuals, a description of the knowledge about the concrete elements in the world. Assertion mining is the search for useful information in this data which is going to be represented as terminological knowledge.

[1]To simplify the presentation we assume from now on that $\mathcal{T}$ is empty or unfoldable.

Example 1 A knowledge base Σ^{arr} for arrhythmia diseases

For the knowledge base $\Sigma^{\text{arr}} = (\mathcal{T}_{\text{arr}}, \mathcal{A}_{\text{arr}})$ there are some simple but non-trivial examples of the reasoning processes:

$$\begin{aligned} \mathcal{A}_{\text{arr}} = \{ \; & \textsc{pat}_1\text{: } \textsc{Male} \sqcap \neg\textsc{Smoker} \sqcap \forall\textsc{hasrelative}.\textsc{Tachycardic}, \\ & (\textsc{pat}_1,\textsc{pat}_1)\text{:}\textsc{hasrelative},\ \textsc{pat}_1\text{:}\forall\textsc{has\_pwave}.\text{OK} \\ & (\textsc{pat}_2,\textsc{pw})\text{:}\textsc{has\_pwave},\ \textsc{pw}\text{:}\neg\text{OK},\ \textsc{pat}_2\text{:}\textsc{Arrhythmia}\ \} \\ \mathcal{T}_{\text{arr}} = \{ \; & \textsc{Tachycardic} \mathrel{\dot\sqsubseteq} \textsc{Arrhythmia} \sqcap \neg\textsc{LowHeartRate}, \\ & \textsc{Hypertrophic} \doteq \textsc{Arrhythmia} \sqcap \exists\textsc{has\_pwave}.\neg\text{OK}\ \} \end{aligned}$$

1. *ABox consistency:* $\Sigma^{\text{arr}} \models \mathcal{A} \neq \bot$ but $\Sigma^{\text{arr}} \models \mathcal{A} \cup \{\textsc{pat}_1\text{:}\neg\textsc{Arrhythmia}\} = \bot$.
2. *Subs.:* $\Sigma^{\text{arr}} \models \forall\textsc{hasrelative}.\textsc{Tachycardic} \sqsubseteq \forall\textsc{hasrelative}.\textsc{LowHeartRate}$.
3. *Instance checking:* $\textsc{pat}_1 \in_{\Sigma^{\text{arr}}} \textsc{Arrhythmia}$

For this purpose we consider a supervised learning approach on data which is classified into decision classes. Here classification corresponds to instance checking with respect to a set $\mathcal{D} = \{D_1, \ldots, D_n\}$ of concepts (which we call *decisions*). Each object o in an ABox $\mathcal{A}$ which is an instance of at least one (but possibly more) decisions $o \in_\Sigma D_i$ in $\mathcal{A}$ is called classifiable. Let $class(\mathcal{A})$ denote the set of all classifiable objects in $\mathcal{A}$. In Σ^{arr} the ABox objects $\textsc{pat}_1$ and $\textsc{pat}_2$ are classifiable w.r.t. $\mathcal{D}_{\text{arr}} = \{\textsc{Tachycardic}, \textsc{Hypertrophic}\}$, $\textsc{pw}$ is not.

ABox mining is now the search for a formal definition for each of the decisions in $\mathcal{D}$ which might eventually be added to the TBox after being evaluated and assessed. In the process of the generalisation of a decision D_i, we will call the instances of D_i in $\mathcal{A}$ the *positive examples* and all instances of the remaining decisions *negative examples.*

Learning Criteria: Generalised Decision Concepts Whether a learned concept L_D can be used as a formal definition of a decision D depends on the knowledge which is represented in the ABox $\mathcal{A}$, the TBox $\mathcal{T}$ and on some additional *learning criteria.* The primary criterion we identified in [12] is exclusiveness. A concept L_D is *exclusive* w.r.t. a decision D if for all objects a in $class(\mathcal{A})$: $a \notin_\Sigma D \Rightarrow a \in_\Sigma \neg L_D$. Exclusiveness implies "correctness" of classification w.r.t. the original decision, i.e. $\forall a \in_\Sigma L_D \Rightarrow a \in_\Sigma D$, a condition traditionally known as *covering.* Furthermore, exclusiveness garanties coveredness even for the case that more information about elements in the ABox become available. Given the Open World Assumption which usually underlies description logic based knowledge representation, this seems to be a crucial requirement. Based on Rough Set theory [10] we also show in [12] that exclusiveness provides a theoretical notion of *safe data* which ensures good properties w.r.t. noisy data.

A learned concept L_D should also be supported by the fact that there is at least one example in the ABox, which is an instance of L_D and we define a *witness* for L_D as an individual $a \in D$ in $class(\mathcal{A})$ such that $a \in_\Sigma L_D$.

Definition 1.1 *A* generalised decision concept (GDC) *for a decision D with respect to a knowledge base $\Sigma = (\mathcal{T}, \mathcal{A})$ and a set of decisions $\mathcal{D}$ is a concept, which is exclusive w.r.t. D and $\mathcal{D}$ and for which a witness exists in $\mathcal{A}$.*

Given the definition of exclusiveness, the set of all GDCs for a decision D_i is:

$$\begin{aligned}\underline{G}(D_i) = \{C \in \mathcal{DL} \mid\ & \exists o \in \mathcal{A} : o \in_\Sigma C \text{ and} \\ & \forall a \in \mathcal{A} : a \notin_\Sigma D_i \Rightarrow \forall_{k \neq i} D_k \in \mathcal{D}\ (a \in_\Sigma D_k \Rightarrow a \in_\Sigma \neg C).\end{aligned}$$

There are several GDCs for the decision Tachycardic w.r.t. Σ^{arr} among them $\forall$has_pwave.OK and $\forall$has_pwave.OK $\sqcup \neg$Arrhythmia.

Since there is still a possibly infinite number of GDCs we need to identify an inductive bias to be able to formally define the necessary restriction of the hypothesis space.

Inductive Bias: Common Vocabulary and Polarity. There are different aspects related to the choice of the inductive bias underlying our approach and we will discuss them briefly.

Object abstraction. We use the fact that DLs have language features (quantification) to reason about objects in an abstract way, generalising from particular examples to more general facts (e.g. existence of a role successor).

Common language with Discerning polarity. We take the designer of the knowledge base and the witnesses they provided seriously, and relate the learned concepts to the vocabulary of each example. The GDCs should consist of the vocabulary which is both used in the examples and the counterexamples and which is used in the same way (i.e. with the same polarity) as it was in the examples. Consider the GDCs for Tachycardic as given above. The second one contains the negated atomic concept Arrhythmia. But there is no reason to assume that people without arrhythmia are likely to have a Tachycardic condition. Since we know on the other hand that all of pat_1's p_waves are o.k as opposed to pat_2's it seems to be a legitimate inductive leap to assume that $\forall$has_pwave.OK is a *good* GDC for the decision Tachycardic. This bias is usually not discussed in the learning literature, because it automatically applies to most learning methods or is enforced using algorithms for missing attribute values.

Maximality. In addition to these syntactical restrictions we have to discuss the semantical choice of GDCs according to the subsumption ordering. In assertion mining the search space for possible learned concepts is defined by comparison of examples with counterexamples which is then restricted through the syntactical bias as described above. Since the learned concepts are defined explicitely as to exclude the counterexamples it seems appropriate to choose the maximal such concepts. This is a more or less arbitrary choice[2] which is related

[2]The choice is arbitrary because you can either minimise the description of the positive or of the negative examples. The first choice corresponds to the least common subsumer approach,

to ongoing research to deconstruct GDCs into sub-GDCs as an evaluation procedure. Such subconcepts of the GDCs will simplify the constructed GDCs and can be used to fine-tune the required level of generalisation.

It has to be mentioned that the algorithms presented in this paper do *not* calculate maximal GDCs w.r.t. the subsumption hierarchy. For technical reasons related to the definition of the common language of two examples we calculate *big* but not *maximal* concepts. For a more detailed discussion we refer to [12].

In the remaining sections we will introduce interpolation for hybrid knowledge representation systems and show how to use algorithms for interpolation to calculate such big GDCs with common vocabulary.

2 Interpolation Methods for Assertion Mining

Having defined learning targets and an inductive bias to restrict the hypothesis space it remains to provide effective algorithms to find some of these GDCs. We sketch the mining process from input (a knowledge base and a set of decisions) to the final output of new terminological axioms in the following section. This process is based on ABox interpolation. Vaguely speaking, interpolants are intermediate formulas semantically linking two other formulas using common vocabulary. We extend this notion to incorporate implicit knowledge about objects and polarity. The idea for the definition of the common language is based on the traditional notion of Lyndon interpolation for first order logic [9].

The language $occ(C)$ is a set of concept names occurring in C, labelled with the quantifier depth. A concept name A has positive (negative) polarity if it is embedded in an even (odd) number of negations. The quantifier depth describes the sequence of roles the concept is quantified over. $\tilde{occ}(C)$ is a dual notion in the sense that $\tilde{occ}(C) = occ(\neg C)$. Formally:

$$\begin{array}{ll}
occ(C)^s = \{(A,+)^s\},\ \tilde{occ}(C)^s = \{(A,-)^s\} & \text{if } C = A \text{ and } A \text{ is an atom.}\\
occ(C)^s = \tilde{occ}(D)^s,\ \tilde{occ}(C)^s = occ(D)^s & \text{if } C = \neg D\\
occ(C)^s = occ(C_1)^s \cup occ(C_2)^s & \text{if } C = C_1 \sqcap C_2 \text{ or } C = C_1 \sqcup C_2\\
\tilde{occ}(C)^s = \tilde{occ}(C_1)^s \cup \tilde{occ}(C_2)^s & \text{if } C = C_1 \sqcap C_2 \text{ or } C = C_1 \sqcup C_2\\
occ(C)^s = occ(D)^{sR} & \text{if } C = \exists R.D \text{ or } C = \forall R.D\\
\tilde{occ}(C)^s = \tilde{occ}(D)^{sR} & \text{if } C = \exists R.D \text{ or } C = \forall R.D
\end{array}$$

Based on this language we define concept interpolation, the basis for ABox interpolation. From now on we will use the letters I and L for (Lyndon) interpolants. *Concept interpolation* holds for two concepts C and D where $C \sqsubseteq D$ if there is a concept I such that $C \sqsubseteq I$ and $I \sqsubseteq D$ and $occ(I) \subseteq occ(C) \cap occ(D)$.

ABox interpolants connect the implicit positive knowledge about an objects a with the negative information about an object b, thus highlighting the differences

the second to assertion mining.

between them. A concept I is an *ABox interpolant* for a and b iff $a \in_\Sigma I$ and $b \in_\Sigma \neg I$ and where $occ(I) \subseteq occ^{\mathcal{A}}(a) \cap \tilde{occ}^{\mathcal{A}}(b)$. The language for an instance a is the smallest set $occ^{\mathcal{A}}(a)$ such that $occ(C) \subseteq occ^{\mathcal{A}}(a)$ for all conceptual axioms $(a : C) \in \mathcal{A}$, $occ^{\mathcal{A}}(b)^R \subseteq occ^{\mathcal{A}}(a)$ for all $(a, b) : R \in \mathcal{A}$ and $occ(C) \subseteq occ^{\mathcal{A}}(a)$ if $(b, a) : R \in \mathcal{A}$ and $b : \forall R.C \in \mathcal{A}$. $\tilde{o}cc^{\mathcal{A}}(a)$ is defined similarily.

Partial ABox interpolation plays an intermediate role between concept and ABox interpolation and relates knowledge about an object with a concept. A concept L is a *partial interpolant* for an object o, an ABox $\mathcal{A}$ and a concept C where $o \in_\Sigma C$ if and only if: $C \sqsubseteq L$ & $o \in_{\mathcal{A}} \neg L$ and $occ(L) \subseteq \tilde{occ}^{\mathcal{A}}(o) \cap occ(C)$.

Partial interpolation for an ABox $\mathcal{A}$ can be reduced to concept interpolation by construction of a preprocessing complete ABox $\mathcal{A}'$ for $\mathcal{A}$ [6]. Furthermore ABox interpolation can be reduced to partial interpolation by propagation of some properties of objects into the ABox.

2.1 Assertion Mining and Interpolation

It is easy to see that the set of ABox interpolants for a classified object $a \in_\Sigma D$ (the examples for D in $\mathcal{A}$) with all other classified individuals $b \notin_\Sigma D$ (the counterexamples) coincides with the set of generalised decision concepts.

Theorem 2.1 *If there are any classified individuals for a decision D and a knowledge base Σ, $\underline{G}(D)$ is the set of all ABox interpolants for the classified objects $a \in_\Sigma D$ with respect to the set of all classified instances $b \notin_\Sigma D$.*

Every generalised decision concept C for a decision D is an ABox interpolant for an $a \in_\Sigma D$ and all $b \notin D$ because there is a witness and because $b \in_\Sigma \neg C$ for all $b \notin_\Sigma D$. But every ABox interpolant I for a and all b obviously instantiates every b into $\neg I$ and since $a \in_\Sigma I$ is a witness, I is a generalised decision concept.

This theorem is the theoretical foundation for Assertion Mining using interpolation methods. The idea to use interpolants for assertion mining is simple: the input is an ABox $\mathcal{A}$ and a set of decisions $\mathcal{D}$ and an empty or unfoldable TBox. Preprocessing of the ABox might consist of unfolding, restriction to fewer elements, retrieval of the instances of each decision, etc. For each decision D in $\mathcal{D}$ a generalised decision concept for each decision (if defined) is calculated as the disjunction over the GDCs for each possible witness. The GDCs for each witness are just the conjunctions over the ABox interpolants of the witness with all the negative examples. The evaluation of the GDCs could comprise statistical analysis, decomposition into minimal subconcepts or involve human expertise.

Algorithm 2.1 summarizes the assertion mining procedure in pseudo code where `ABox_LI`$(\mathcal{A}, a, b)$ is a method to calculate an ABox interpolant for two elements a and b with respect to an ABox $\mathcal{A}$. Such a procedure for $\mathcal{ALC}$ will be presented in the following section based on tableau calculi.

Algorithm 2.1 `disc_aboxmine`$(\mathcal{A}, \mathcal{D})$ Discernibility based Assertion Mining

Input: An ABox $\mathcal{A}$ and a set of decisions $\mathcal{D}$.
Output: A TBox $\mathcal{T} = \bigcup_{D \in \mathcal{D}} \{D \doteq G_D\}$, where all G_D are "almost" maximal GDCs.

```
$\mathcal{A}^* :=$ preprocess$(\mathcal{A})$;
$\mathcal{T} := \varnothing$;
for all $D \in \mathcal{D}$
    for all $a \in class(\mathcal{A}^*)$
        if $a \in D$
           for all $b \in class(\mathcal{A}^*)$
               if not $b \in D$
                  if ABox_LI$(\mathcal{A}^*, a, b)$ is defined
                     $LI^a := LI^a \sqcap$ ABox_LI$(\mathcal{A}^*, a, b)$;
    $GDC := GDC \sqcup LI^a$;
    evaluate(GDC);
    $\mathcal{T} := \mathcal{T} \cup \{D \doteq GDC\}$;
return $\mathcal{T}$;
```

2.2 Tableau Methods for Interpolation.

The algorithms to calculate interpolants using logical tableaux presented here follow the lines of [7]. Interpolants for $\mathcal{ALC}$ concept subsumption can be constructed from a fully expanded closed tableau collecting contradicting literals on each branch using construction rules corresponding to traditional tableaux. For ABox interpolants the more complex interaction between role and concept assertions has to be taken into account. The solution is to preprocess the ABox (according to [6]) and propagate the result of all possible inference steps into a concept. This concept can then be used in an intermediate interpolation step to represent the complete knowledge about one of the objects. Application of the same preprocessing steps then allow a further reduction of the interpolation problem to concept interpolation.

The algorithms described in this paper are based on tableau proof calculi. A tableau is a set of branches, where each branch is a set of formulas. A formula is a term of the form $(a : C)$ or $(a, b) : R$ where a and b are individual variables, C an $\mathcal{ALC}$ concept and R a role name. A branch is closed if it contains two formulas $a : C$ and $a : \neg C$. The notions of open branch and closed and open tableau are defined as usual. We will identify a branch B with the set of formulas ϕ on the branch and write $\phi \in B$. To simplify the presentation of our rules, we assume that all formulas are in *negation normal form*, i.e. negation is always pushed down to the atomic level [13]. The algorithms to calculate ABox interpolant are based on concept interpolation. Since concept interpolation is similar to interpolation in modal logic K and because of the close connection between $\mathcal{ALC}$ and K [11] we omit further details and assume that there is a procedure concept_LI(C_1, C_2) which calculates a concept interpolant for C_1 and C_2. Please consult [12, 7] for further details.

($\sqcap$):	**if**	$(a : C_1 \sqcap C_2) \in B$, but not both $(a : C_1) \in B$ and $(a : C_2) \in B$
	then	$B' := B \cup \{(a : C_1), (a : C_2)\}$.
($\sqcup$):	**if**	$(a : C_1 \sqcup C_2) \in B$, but neither $(a : C_1) \in B$ nor $(a : C_2) \in B$.
	then	$B' := B \cup \{a : C_1\}$ and $B'' := B \cup \{a : C_2\}$.
($\forall$):	**if**	$(a : \forall R.C) \in B$ and $((a, b) : R) \in B$ but not $(b : C) \in B$.
	then	$B' := B \cup \{b : C\}$.

Figure 1: Preprocessing Rules

Algorithms for ABox interpolation. To calculate ABox interpolants we adapt the reduction introduced by Hollunder in [6]. In order to reduce ABox interpolation to concept interpolation and partial ABox interpolation, we have to preprocess the ABox using the rules in Fig. 1 until no more rule can be applied. In this case the ABox is called preprocessing complete (ppc). Each application of a rule triggers a construction rule for ABox Interpolants defined in Algorithm 2.2, where the concepts B_a and B_c defined in Definition 2.2 and inconsistent(C) is the standard procedure for concept consistency.

Algorithm 2.2 ABox_LI(B, a, b): ABox Lyndon Interpolation

Input: A branch B and two individuals a and b.
Output: ABox Lyndon interpolant for a and b.

```
apply(rule,$B$);
if rule = $(\sqcap), (\forall)$
  $\{B'\}$ := get_new_branches;
  return ABox_LI($B', a, b$);
if rule = $(\sqcup)$
  individual := get_individual;
  $\{B', B''\}$ := get_new_branches;
  if individual = $b$ return ABox_LI($B', a, b$) $\sqcap$ ABox_LI($B'', a, b$);
  else                return ABox_LI($B', a, b$) $\sqcup$ ABox_LI($B'', a, b$);
if rule = undefined;
  if   inconsistent($B_b$) return $\top$;
  if   there is an individual name $c \neq b$ such that inconsistent($B_c$) return $\perp$;
  else return propagate_$LI$($B, a, b$);
```

Hollunder shows that for any consistent ABox $\mathcal{A}$ there is a consistent ppc ABox $\mathcal{A}'$ derivable from $\mathcal{A}$. We identify $\mathcal{A}'$ with the set of all branches B build from $\mathcal{A}$. An ABox $\mathcal{A}$ is consistent if there is an open branch B in the ppc ABox $\mathcal{A}'$. For each of these open branches we now construct interpolants as defined in Algorithm 2.3. For this purpose we explicitely collect the information about an object a on a branch B into a concept B_a as defined below.

Definition 2.2 *The set $\mathcal{C}_a^B$ of concepts related to an individual $a \in B$ in a branch B is defined as follows: $C \in \mathcal{C}_a^B$ iff*

- *$a : C \in B$, where C is a literal, i.e. a concept name or its negation,*
- *$C = \exists R.D$ and $a : \exists R.D \in B$ if there is no R successor of a in B or*
- *$C = \forall R.D$ and $a : \forall R.D \in B$.*

For an individual $a \in B$ and a branch B we define a concept $B_a = \bigsqcap_{C \in \mathcal{C}_a^B} C$.

The concept B_a now contains all the conceptual information about an object a in B and an interpolant for two objects a and b on a branch B is simply the concept interpolant for B_a and $\neg B_b$. But there might still be interpolants relating a role assertion $(a, c) : R \in B$ with a universal quantified statement $b : \forall R.C$.[3] But this interpolant is just the R–quantified negated partial interpolant for C and c and to construct "almost" maximal interpolants, all possible partial interpolants of this kind are added disjunctively. To make sure that all interactions between role successors are detected, we have to propagate simultaniously.

Algorithm 2.3 `propagate_LI`(B, a, b): Propagation for ppc ABoxes

Input: A ppc consistent branch B and two individuals a and b.
Output: ABox Lyndon Interpolant for a and b with respect to B.

```
for all assertions $((b,d_1) : R) \in B$ to $((b,d_n) : R) \in B$
    for the set of all formulas $\{a : \forall R.C_1, \dots, a : \forall R.C_m\} \subseteq B$
                                which are universally quantifying over role $R$ for $a$
        if partial_LI$(B, C_1 \sqcap \dots \sqcap C_m, \{d_1, \dots, d_n\})$ exists
          $LI := LI \sqcup \forall R.$partial_LI$(B, C_1 \sqcap \dots \sqcap C_m, \{d_1, \dots, d_n\})$;
for all assertions $((a,c_1) : R) \in B$ to $((a,c_n) : R \in B$
    for the set of all formulas $\{b : \forall R.C_1, \dots, b : \forall R.C_m\} \subseteq B$
                                which are universally quantifying over role $R$ for $b$
        if partial_LI$(B, C_1 \sqcap \dots \sqcap C_m, \{c, \dots, c_n\})$ exists
          $LI := LI \sqcup \exists R.\neg$partial_LI$(B, C_1 \sqcap \dots \sqcap C_m, \{c_1, \dots, c_n\})$;
if concept_LI$(B_a, \neg B_b)$ exists
  $LI := LI \sqcup$ concept_LI$(B_a, \neg B_b)$;
return $LI$;
```

For partial interpolation for a set A of object a_1 to a_n and a concept C we again have to preprocess the branch and to expand the concept C until all possible interactions are made explicit. Algorithm 2.4 describe the preprocessing steps, Algorithm 2.5 the expansion of the concept into sets of concepts.

Note that the expansion of the concepts $C \in S$ can recursively trigger partial interpolation. If all rules are exhaustively applied the partial interpolant is

[3]If e.g. $c : \neg C \in B$, $\exists R.\neg C$ is an ABox interpolant for a and b w.r.t. B.

Algorithm 2.4 `partial_LI`(B, C, A): Disjunctive Partial Interpolation

Input: A consistent Branch B, a concept C and individuals $A = \{a_1, \dots, a_n\}$
Output: Disjunctive Partial LI L: $C \sqsubseteq L$ and there is an $a_i \in_B \neg L$ for $1 \leq i \leq n$ and $occ(L) \subseteq \tilde{occ}^B(a_i) \cap occ(C)$

```
apply(rule,$B$);
if rule = ($\sqcup$)
  $\{B', B''\}$ := get_new_branches;
  return partial_LI($B', C, \{a_1, \dots, a_n\}$) $\sqcap$ partial_LI($B'', C, \{a_1, \dots, a_n\}$);
if rule = ($\sqcap$), ($\forall$)
  $\{B'\}$ := get_new_branches;
  return partial_LI($B', C, \{a_1, \dots, a_n\}$);
if rule = undefined;
  return exp_concepts($B, \{C\}, \{a_1, \dots, a_n\}$);
```

just the concept interpolant for the conjunction over the concepts in S and the disjunction of the negations of the B_a as defined in Definition 2.2 for all $a \in A$.

Algorithm 2.5 `exp_concepts`(B, S, A): Partial Interpolation

Input: A ppc branch B, a set of concepts S and individuals $A = \{a_1, \dots, a_n\}$.
Output: Disjunctive partial LI: $\sqcap_{C \in S} C \sqsubseteq L$ and there is an $a_i \in_B \neg L$, $occ(L) \subseteq \tilde{occ}^B(a_i) \cap occ(C)$.

```
if there is a concept $C_1 \sqcap C_2$ such that $S = S' \cup \{C_1 \sqcap C_2\}$
  return exp_concepts($B, S' \cup \{C_1\} \cup \{C_2\}, A$);
else if there is a concept $C_1 \sqcup C_2$ such that $S = S' \cup \{C_1 \sqcup C_2\}$
    return exp_concepts($B, S' \cup \{C_1\}, A$) $\sqcup$ exp_concepts($B, S' \cup \{C_2\}, A$);
else if for all assertions $(a_i, b_i) : R \in B$ where $a_i \in A$
    for all universally quantified concepts $\forall R.C_1, \dots, \forall R.C_n \in S$ over $R$
        return exp_concepts($B, S, A$) $\sqcup \forall R.$ partial_LI($B, C_1 \sqcap \dots \sqcap C_n, \bigcup_i \{b_i\}$)
else return concept_LI($\sqcap_{C \in S} C, \neg B_{a_1} \sqcap \dots \sqcap \neg B_{a_n}$);
```

In [12] soundness and completeness of the algorithms is shown in the sense that whenever there is one, `ABox_LI`$(\mathcal{A}, a, b)$ returns with an ABox interpolant for a and b w.r.t $\mathcal{A}$.

Let us consider the simple ABox $\mathcal{A} = \{(a, a) : R, a : C, b : \forall R.\forall R.\neg C\}$ which is preprocessing complete. To calculate an ABox interpolant for a and b we call `propagate_LI`$(\mathcal{A}, a, b)$ which returns $\exists R.\neg$`partial_LI`$(\mathcal{A}, \forall R.\neg C, \{a\})$. $\mathcal{A}$ is still ppc and `exp_concepts`$(\mathcal{A}, \{\forall R.\neg C\}, \{a\})$ returns $\forall R.$`partial_LI`$(\mathcal{A}, \neg C, \{a\})$, which is simply $\forall R.$`concept_LI`$(\neg C, \neg C)$. But this is $\forall R.\neg C$ and the ABox interpolant for a and b is therefore just $\exists R.\neg \forall R.\neg C$, which is $\exists R.\exists R.C$.

2.3 Assertion Mining and Most Specific Concepts

Our algorithms avoid the construction of a representation for the ABox objects and therefore of the most specific concepts[4]. This is possible due to exclusiveness, because all counterexamples are required to be instances of the negation of all GDCs. This condition is much stronger than the one used for lcs–learning, where counterexamples are only required not to be instances of the learned concepts. ABox interpolation captures this notion of instance checking because it corresponds to a search for possible contradictions between properties of positive and negative examples.[5] But in $\mathcal{ALC}$ the set of possible contradictions in an ABox is finite even if ABox cycles exist. Note however, that the algorithms described above do not terminate for description logics with number restrictions or features because this would allow for dual contradicting infinite chains. In this case non-maximal approximations of GDCs have to be constructed and the method will not be complete.

2.4 Practical Experience

Wellington' Kat implements interpolation based assertion mining as an extension of the hybrid knowledge representation system Wellington, which was developped at King's College London [4]. First tests have been performed on a knowledge base Σ^{arr} for cardiac arrhythmia [5], which contains various information including ECG data (based on 279 attribute values) about 452 patients, which are classified into 16 classes of cardiac arrhythmias. Σ^{arr} has been created by translation from a database available at [2] and has maximal relational depth of three. First results indicate that assertion mining without evaluation procedures over-generalises, i.e. too many previously unknown examples are instances of too many GDCs. Another crucial problem is related to efficiency and complexity. Although the learning procedure itself is reasonably fast, the GDCs which are constructed are too complex for efficient subsequent instance checking of new objects. Both these issues are related and we are currently implementing evaluation procedures to simplify GDCs which include symbolic (e.g. logical decomposition) as well as statistical methods.

3 Conclusion

We have presented algorithms for assertion mining based on interpolation which calculate generalised decision concepts even if the most specific concept for some objects in the ABox do not exist. They allow for the construction of new termi-

[4]The *most specific concept* $msc(a)$ for an ABox instance a is a minimal concept in the set of concepts which instantiate a, i.e. $a \in_\Sigma C$ implies $\Sigma \models msc(a) \sqsubseteq C$ for all $C \in \mathcal{ALC}$.

[5]In an $\mathcal{ALC}$ ABox $\mathcal{A}$, an ABox individual a is an instance of a concept C if and only if $\mathcal{A} \cup \{a : \neg C\}$ is inconsistent.

nological knowledge from classified ABox instances. We are currently extending the DL based WELLINGTON representation and reasoning system with a KAT, a knowledge acquisition tool based on the presented methods.

References

[1] F. Baader and R. Küsters. Least common subsumer computation w.r.t. cyclic $\mathcal{ALN}$-terminologies. In *Proceedings of the 1998 International Workshop on Description Logics (DL'98)*, 1998.

[2] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[3] W. Cohen and H. Hirsh. Learning the classic description logic: Theoretical and experimental results. In *KR-94*, pages 121–133, Bonn, Germany, 1994.

[4] U. Endriss. Reasoning in description logic with WELLINGTON 1.0. In *Proceedings of the Automated Reasoning Workshop 2000*, London, UK, 2000.

[5] H.A. Güvenir, B. Acar, G. Demiröz, and A. Cekin. A supervised machine learning algorithm for arrhythmia analysis. In *Computers in Cardiology*, volume 24, pages 433–436, 1997.

[6] B. Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Annals of Mathematics and Artificial Intelligence*, 18:95–131, 1996.

[7] M. Kracht. *Tools and Techniques in Modal Logic*. North Holland, 1999.

[8] R. Küsters and R. Molitor. Computing most specific concepts in description logics with existential restrictions. LTCS-Report 00-05, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 2000.

[9] R.C Lyndon. An interpolation theorem in the predicate calculus. *Pacific Journal of Mathematics*, 9:155–164, 1959.

[10] Z. Pawlak. Rough sets. *International Journal of Computer and Information Sciences*, 11(5):341–356, 1982.

[11] Klaus Schild. A correspondence theory for terminological logics: preliminary report. In *Proceedings of IJCAI-91, 12th International Joint Conference on Artificial Intelligence*, pages 466–471, Sidney, AU, 1991.

[12] S. Schlobach. Interpolation methods for assertion mining in hybrid knowledge bases. Technical report, King's College London, 2001.

[13] M. Schmidt-Schauss and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

Ontology Language Integration: A Constructive Approach

Heiner Stuckenschmidt[1] **and Jérôme Euzenat**[2]

[1]Center for Computing Technologies,University of Bremen
email: heiner@tzi.de

[2]INRIA Rhônes-Alpes, Grenoble
email: Jerome.Euzenat@inrialpes.fr

Aug 24th 2001

Abstract

The problem of integrating different ontology languages has become of special interest recently, especially in the context of semantic web applications. In the paper, we present an approach that is based on the configuration of a joint language all other languages can be translated into. We use description logics as a basis for constructing this common language taking advantage of the modular character and the availability of profound theoretical results in this area. We give the central definitions and exemplify the approach using example ontologies available on the Web.

1 Motivation

It has been widely recognized that information systems benefit from the use of formal ontologies. These ontologies are used to conceptualize and structure information as well as to provide intelligent search facilities and integration methods for remote information. While ontologies per se already support conceptualization and structuring, applications like intelligent search and information integration make it possible to reason about the knowledge specified in the ontologies. This requirement, in turn requires the ontology to be implemented in a machine processable language. Thus, the question of using ontologies in information systems is also a question of the language used to encode the ontologies.

The World Wide Web is the largest information system ever. Its size and heterogeneity makes ontology based search and integration even more important than in other information systems. In this context the “semantic web” [Berners-Lee et al., 2001] is mentioned. It is supported by the annotation of web pages, containing informal knowledge as we know it now, with formal knowledge. These documents can reference each other and depend on background knowledge. Taking advantage of the semantic web requires to be able to gather, compare, transform and compose the annotations. For several reasons (legacy knowledge, ease of use, heterogeneity of devices and adaptability, timelessness), it is not likely that this formal knowledge will be encoded in the very same language. The interoperability of formal knowledge languages must then be studied in order to interpret the knowledge acquired through the semantic web.

2 Language Construction

In the words of Tim Berners-Lee, the semantic web requires a set of languages of increasing expressiveness and anyone can pick up the right language for each particular semantic web application. This is what has been developed by the description logic community over the years. Our approach focuses on ontology languages that rely on these logics. The rationale for this choice is the following:

- The expressiveness and complexity of these languages has been studied thoroughly and well-founded results are available [Donini et al., 1991],[Donini et al., 1994]
- It has been shown that description logics provide a unifying framework for many class-based representation formalisms [Calvanese et al., 1999].
- Description logic-based languages have become of interest in connection with the semantic web. the languages OIL [Fensel et al., 2000] and the DAML language [McGuinness et al., 2001] are good examples.

A modular family of languages is a set of languages made from a set of operations (constituting an algebraic base) that can be combined. Since the languages have a similar kind of semantic characterization, it is easier to transform a representation from one language to another and one can take advantage of efficient provers or expressive languages.

2.1 Customized Languages

Relying on description logics we already get a notion of a special language from the combination of operators. Theoretical results from the field of description logics provide us with the knowledge about decidable combinations of modeling primitives and their complexity with respect to subsumption reasoning. Consequently, every decidable combination of operators is a potential pattern that can be used to build the ontology for a certain application. In the course of the engineering process we have to handle different language patterns:

Reasoner Languages the languages that available reasoners are able to handle.

Legacy Languages the language a useful, already existing ontology is encoded in.

Acquisition Languages are languages needed to encode acquired knowledge.

The Goal Language describes a language that can act as an interlingua for the ontologies to be integrated. It represents a trade-offs between expressivity constraints of the legacy and acquisition languages and the complexity constraints of the reasoner languages.

In order to find the goal language, we have to find an optimal trade-off between the other languages involved. For this purpose we invent the notion of coverage for languages. A language L' is said to cover a language L, if there is a transformation from L to L' that preserves consequence. In particular, this is the case if all modeling primitives from L are also contained in L' or can be simulated by a combination of modeling primitives from L'. We denote the fact that L' covers L as $L \prec L'$. Using the notion of coverage we can now define the customization task.

Definition: Customization Task. A customization task is defined by a triple $\langle \mathcal{R}, \mathcal{U}, \mathcal{A} \rangle$ where $\mathcal{R}$ is a set of reasoner languages, $\mathcal{U}$ a set of legacy languages and $\mathcal{A}$ a set of acquisition languages. The language G is a solution of the customization task if it is a language that is covered by a reasoner language and covers all reuse and acquisition languages, or formally:

$$\exists R \in \mathcal{R}(G \prec R) \wedge \forall P \in \mathcal{U} \cup \mathcal{A}(P \prec G) \tag{1}$$

This definition provides us with an idea of the result of the customization process. However there are still many technical and methodological problems. We have to investigate the nature of the covering predicate and develop an algorithm for deciding whether one pattern covers the other. We introduce different notions of coverage of increasing strength that is based on transformations in the next section.

2.2 A Transformation-Based Approach

The notion of transformability is a central one in our approach because it allows to define the coverage relation. The simplest transformation is the transformation from a logic to another which adds new constructors. The transformation is then trivial, but yet useful, because the initial representation is valid in the new language.

In the following we use L and L' to refer to languages. Languages are sets of expressions δ. Representations r are sets of expressions which are normally subsets of a language. $r_L \subseteq L$. Transformations are mappings $\tau : L \rightarrow L'$ from expressions in one language to expressions in a different language.

Definition: Syntactic Coverage This trivial form of transformation provides us with a first notion of coverage that reflects the situation, where a language is the subset of the other:

(2) $$L \overline{\ll} L' \Leftrightarrow_{def} (L \subseteq L')$$

For this case, one can define a special case of the coverage relation as $L \overline{\ll} L'$ which means that one language is completely included in the other in a lexical sense.

If $L \overline{\not\ll} L'$, the transformation is more difficult. The initial representation r can be restricted to what is (syntactically) expressible in L'. However, this operation (which is correct) is incomplete because it can happen that a consequence of a representation expressible in L' is not a consequence of the expression of that representation in L'. The preceding proposal is restricted in the sense that it only allows in the target language, expressions expressible in the source language, while there are equivalent non-syntactically comparable languages. This is the case of the description logic languages $\mathcal{ALC}$ and $\mathcal{ALUE}$ which are known to be equivalent while being defined by different operators. For that purpose, one can define $L \overline{\overline{\ll}} L'$ if and only if the models are preserved.

Definition: Semantic Coverage Transformations that simulate some operators of the transformed language using combinations of operators of the goal language imply a notion of coverage that is based on the semantics of languages. Let I be the a Tarskian style interpretation defining the model-theoretic semantics of expressions, then we get

(3) $$L \overline{\overline{\ll}} L' \Leftrightarrow_{def} \forall I : I \models_L \delta \Rightarrow I \models_{L'} \tau(\delta)$$

Another possibility is to define $\widetilde{\ll}$ as the existence of an homomorphism between the models of the original and the translated language. This property guarantees that inconsistency of an expression in the target language implies inconsistency of the expression in the source language.

Definition: Model-Theoretic Coverage Transformations that preserves inconsistency which is an important property with respect to automated reasoning by guaranteeing that for every model in L there also is a model in L' define a special case of semantic coverage we refer to as model-theoretic coverage:

$$L \widetilde{\ll} L' \Leftrightarrow_{def} \forall I \exists I' : I \models_L \delta \Rightarrow I' \models_{L'} \tau(\delta) \tag{4}$$

Summarizing, the syntactic and semantic structure of a language family provides us with different criteria for coverage all based on the notion of transformability. These notions of coverage do not only give us the possibility to identify and prove coverage, it also specifies a mechanisms for transforming the covered into the covering language. Therefore we not only show that a suitable language can be generated, but also how the generation is being performed. In the next section we present an implementation of this approach.

3 Transformation-based Ontology Integration

The notion of semantic interoperability is a very broad one since it covers almost all application of the semantic web. Therefore we can only give evidence for the usefulness of the 'family of languages' approach by example.

3.1 An Example Problem

We chose a scenario where two existing ontologies should be integrated to establish a semantic model of an application domain. The library of the DAML (DARPA Agent Markup Language) contains an ontology describing a technical support application (http://www.daml.org/ontologies/69). It is encoded in the DAML-ONT language.

```
<Class ID="ProductInfo">
    <subClassOf
     resource="#IncommingTechSupIncident"/>
    <comment>
       Technical Product Information
    </comment>
```

```
</Class>

<Property ID="operatingSystem">
    <domain resource="#ProductInfo"/>
    <comment>Product’s Operating System</comment>
    <default resource="MSWindows98"/>
</Property>

<Class ID="OperatingSystem">
    <oneOf parseType="daml:collection">
       <OperatingSystem ID="MSWindows2000"/>
       <OperatingSystem ID="MSWindowsNT"/>
       <OperatingSystem ID="MSWindows98"/>
       <OperatingSystem ID="MSWindows95"/>
    </oneOf>
    <comment>
       Available Operating Systems
    </comment>
</Class>

<Property ID="productVersion">
    <domain resource="#ProductInfo"/>
    <comment>Product’s Version</comment>
    <default resource="#PersonalEdition"/>
</Property>

<Class ID="ProductVersion">
    <oneOf parseType="daml:collection">
       <ProductVersion ID="EnterpiseEdition"/>
       <ProductVersion ID="DeveloperEdition"/>
       <ProductVersion ID="ProfessionalEdition"/>
       <ProductVersion ID="SmallBusinessEdition"/>
       <ProductVersion ID="PersonalEdition"/>
    </oneOf>
    <comment>Available Product Versions</comment>
</Class>
```

Since the DAML language borrows from description logics [McGuinness et al., 2001, Horrocks, 2000] these constructs can easily be mapped on operators available in the description logic markup language DLML. Operators used in this specific ontology are: atomic names, primitive classes, primitive roles, domain restrictions for assigning properties to classes and the one-of operator for defining classes by enumeration.

We assume that the technical support should be extended to include hardware as well. For this purpose definitions of existing hardware products have to be integrated into the ontology. As an example product we use the printer ontology that can be found at http://www.ontoknowledge.org/oil/case-studies/. This ontology in turn is encoded in the OIL language.

```
<oil:DefinedClass rdf:ID="HPLaserJet1100Series">
    <rdfs:subClassOf>
       <oil:And>
          <oil:hasOperand>
             <rdfs:Class
              rdf:about="#HPLaserJetPrinter"/>
          </oil:hasOperand>
          <oil:hasOperand>
             <rdfs:Class
              rdf:about="#PrinterForPersonalUse"/>
          </oil:hasOperand>
       </oil:And>
    </rdfs:subClassOf>
    <oil:hasPropertyRestriction>
       <oil:HasValue>
          <oil:onProperty
           rdf:resource="#PrintingSpeed"/>
          <oil:toClass>
             <rdfs:Class
              rdf:about="#&quot;8 ppm&quot;"/>
          </oil:toClass>
       </oil:HasValue>
    </oil:hasPropertyRestriction>
    <oil:hasPropertyRestriction>
       <oil:HasValue>
          <oil:onProperty
           rdf:resource="#PrintingResolution"/>
          <oil:toClass>
             <rdfs:Class
              rdf:about="#&quot;600 dpi&quot;"/>
          </oil:toClass>
       </oil:HasValue>
    </oil:hasPropertyRestriction>
</oil:DefinedClass>
```

The semantics of the OIL language is completely specified in terms of description logics. Consequently, we can directly map OIL constructs. Operators used in the model are the following: atomic names primitive concepts, primitive roles, existential restrictions on slot values as well as conjunction for multiple inheritance and multiple slot constraints.

3.2 Integrating the Specifications

Using the family of languages approach, we can integrate the two specifications in a three step process. First, we have to analyze the language patterns (i.e. combinations of operators) at hand then we customize a joint language. Finally, we define and implement transformations between the language patterns and the customized language.

Step 1: Identify Language Patterns The languages used in the specifications from our example, i.e. DAML and OIL are legacy language in the sense of the language customization task. As these languages are very expressive, however for our purpose we only have to care about the part of the languages that are really used in the specifications (see last section).

The second kind of language patterns involved are defined by the aim of providing reasoning support for the integrated specifications. A potential reasoner is the FaCT system that supports two different languages, $\mathcal{SHF}$ and $\mathcal{SHIQ}$ [Horrocks et al., 1999]. Figure 1 illustrates the expressiveness of these languages.

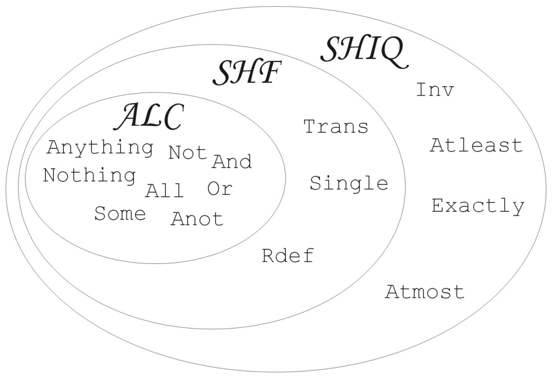


Figure 1: Expressiveness of languages supported by the FaCT reasoner

Step 2: Customize Integration Language The patterns identified in step 1 act as an input for the language customization step. We denote the language pattern used for the technical support ontology as L_{DAML}, the one used for the printer ontology as L_{OIL}. In the example case, we can simply merge the two language patterns into a language that consists of all operators found in both models. The resulting integration language denoted as $L_G = L_{DAML} \overline{\nabla} L_{OIL}$ simply consists of the union of the operators present in the two ontologies described above.

We now have to test the suitability of the pattern. For the suitability we have to check, whether the language covers the legacy languages, i.e. whether $L_G \prec L_{DAML} \wedge L_G \prec L_{OIL}$ holds. In this case this is obvious, because L_G extends both languages ($L_{OIL} \lll L_G$ and $L_{DAML} \lll L_G$). Additionally we have to make sure that L_G is covered by at least one reasoner language (denoted as $L_{\mathcal{SHF}}$ and $L_{\mathcal{SHIQ}}$). We can show that $L_{\mathcal{SHIQ}}$ covers L_G: Concept and role definitions as well as conjunction and existential restriction are already directly contained in $\mathcal{ALC}$ while we have to model `one-of` and `domain` using other operators of the languages supported. This can be done in the very same way as it is done from OIL to the FaCT reasoner [Horrocks, 2000]:

one-of can be simulated using `or`, `not` and `cdef`, because the transformation from $(\texttt{one}-\texttt{of}\ C_1\ C_2)$ to $(\texttt{or}\ C_1\ C_2) \wedge (\texttt{cdef}\ C_1(\texttt{not}\ C_2)))$ preserves inconsistency checking by guaranteeing that for every model for the original expression there is a model in the transformed one. We obtain consequence preservation for this transformation.

domain can be simulated using `all` and `inv`, because

$$R \leq (\texttt{domain}\ C) \Rightarrow \top \leq (\texttt{all}\ (\texttt{inv}\ R)\ C)$$

Another way is to use general concept implications or the form:

$$(\texttt{some}\ R\ \top) \leq C$$

We omit the proofs due to the limited space.

Performing the first transformation we have to use the $\mathcal{SHIQ}$ reasoner, because inverse roles are needed to simulate domain constraints. If we do the second transformation, we can even rely on the $\mathcal{SHF}$ reasoner which supports a smaller language and therefore is able to provide faster reasoning service.

To summarize, we can use the language L_G created by the transformations as a language for the integrated model, because there is a reasoner language (i.e. $L_{\mathcal{SHIQ}}$) that syntactically covers L_G.

Step 3: Implement Languages and Transformations This can be refined in three sub-steps:

1. Translating from DAML and OIL to L_{DAML} and L_{OIL};
2. Providing the transformation from L_{DAML} and L_{OIL} to L_G;

3. Translating from $L_{\mathcal{SHIQ}}$ which syntactically covers L_G to $\mathcal{SHIQ}$.

The implementation has been carried out by transforming representations within the DLML (Description Logic Markup Language [Euzenat, 2001]) framework. It encodes many description logics in XML in a coherent way (same operators have the same name) but does not offer CGI. Transformations are written in the XSLT language for transforming XML documents.

The second one is more related to description logics. It first involves merging both ontologies. This is easily achieved with a straightforward transformation, thanks to the unified vocabulary provided by DLML [Euzenat, 2001](i.e. whatever the logic, the syntax is the same). The resulting logic ($L_{DAML} \bigtriangledown L_{OIL}$) being syntactically stronger than L_{DAML} and L_{OIL} preserves the content of the ontologies as well as the consequence relation.

Then, the resulting merged ontology, which cannot be directly translated into $\mathcal{SHIQ}$ is converted by applying successive transformations (again written in XSLT). The first one eliminates the `domain` constructor and the second one eliminates the `one-of` constructor in exactly the way put forth above. Because the first transformation preserves the models and the second one preserves unsatisfiability, then, the whole chain of transformation preserve consequence.

4 Discussion

We introduced an approach for ontology language integration that is based on the construction of a joint language and the use of semantics-preserving transformations. We outlined the idea of the approach and gave evidence for its suitability using a real-life example.

The approach presented still has several shortcomings implying needs for further research. First of all the nature of different kinds of transformation needs further investigation. We envision a formal framework for proving special properties of transformation in order to guarantee formal properties of the constructed language. When thinking of a web of trust, it is also beneficial to annotation complete proofs to transformed language as a guarantee that no information has been lost.

Another very important related problem which is completely out of the scope of this paper is the problem of translating not only between different representation languages, but also between different terminologies. An approach able to perform translations between different ontologies on the language and the terminology level would be a big step forward.

References

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific Amercan*, 284(5):35–43.

[Calvanese et al., 1999] Calvanese, D., Lenzerini, M., and Nardi, D. (1999). Unifying class-based representation formalisms. *Journal of Artificial Intelligence Research*, 11:1999–240.

[Donini et al., 1991] Donini, F., Lenzerini, M., Nardi, D., and Nutt, W. (1991). The complexity of concept languages. In Sandewall, J. A., Fikes, R., and E., editors, *2nd International Conference on Knowledge Representation and Reasoning, KR-91*. Morgan Kaufmann.

[Donini et al., 1994] Donini, F., Lenzerini, M., Nardi, D., and Schaerf, A. (1994). Deduction in concept languages: from subsumption to instance checking. *Journal of logic and computation*, 4(4):423–452.

[Euzenat, 2001] Euzenat, J. (2001). Preserving modularity in xml encoding of description logics. In McGuinness, D., Patel-Schneider, P., Goble, C., and Mller, R., editors, *Proc. 14th workshop on description logics (DL), Stanford (CA US)*, pages 20–29.

[Fensel et al., 2000] Fensel, D., Horrocks, I., Harmelen, F. V., Decker, S., Erdmann, M., and Klein, M. (2000). Oil in a nutshell. In *12th International Conference on Knowledge Engineering and Knowledge Management EKAW 2000*, Juan-les-Pins, France.

[Horrocks, 2000] Horrocks, I. (2000). A denotational semantics for Standard OIL and Instance OIL. http://www.ontoknowledge.org/oil/downl/semantics.pdf.

[Horrocks et al., 1999] Horrocks, I., Sattler, U., and Tobies, S. (1999). Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, pages 161–180. Springer-Verlag.

[McGuinness et al., 2001] McGuinness, D., Fikes, R., Connolly, D., and Stein, L. (2001). Daml-ont: An ontology language for the semantic web. *IEEE Intelligent Systems*. Submitted to Special Issue on Semantic Web Technologies.