Proceedings of the 2002 Workshop on Applications of Description Logics

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

Aachen, Germany,
September 16, 2002

Abstract

In previous years a growing interest in description logics and their applications was observed. This was 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. Recently, more and more applications of description logics have been developed. In the same spirit as the previous workshop on applications of description logics at KI-2001 this workshop intends to gather researchers as well as practitioners who are interested in description logics and their applications.

Technical Papers

1. Daniela Berardi
   Using DLs to reason on UML class diagrams

2. Sebastian Brandt and Anni-Yasmin Turhan
   An Approach for Optimized Approximation

3. Kerstin Bücher, Günther Görz, and Bernd Ludwig
   Corega Tabs: Incremental Semantic Composition

4. Michael Eisfeld
   Model Construction for Configuration Design

5. A. Huber, G. Görz, R. Zimmermann, S. Käs, R. Butscher, F. Bodendorf
   Design and Usage of an Ontology for Supply Chain Monitoring

6. Chan Le Duc and Nhan Le Thanh
   Approximation from a Description Logic with Disjunction to Another Without Disjunction

7. Bernd Ludwig, Kerstin Bücher, and Günther Görz
   Corega Tabs: Mapping Semantics onto Pragmatics

8. Ragnhild Van Der Straeten and Miro Casanova
   The Use of DL in Component Libraries - First Experiences

Using DLs to reason on UML class diagrams

Daniela Berardi
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
Email: berardi@dis.uniroma1.it

Abstract

UML is the most widely accepted formalism for the analysis and design of software and one of its most important components are UML class diagrams. In this paper we discuss how to encode UML class diagrams in the Description Logics $\mathcal{DLR}_{ifd}$ and $\mathcal{ALCQI}$: the first *fully captures* the semantics of UML class diagrams, while the second is directly supported by state-of-the-art DL reasoning systems. We also show some results obtained by reasoning on UML class diagrams of industrial interest.

1 Introduction

UML is the most widely accepted formalism for the analysis and design of software. One of its most important components are UML class diagrams, which model the static relationships that hold between the objects of the domain of interest, in terms of classes and associations between them. During the design phase, it is highly desirable to be able to detect relevant formal properties of the diagram, such as inconsistencies and redundancies. This requires a formalization of UML class diagrams and several types of formalizations have already been proposed in the UML literature [12, 13, 14, 11].

Description Logics (DLs) [1] are known to be able to capture several conceptual data models [3, 10, 4, 9, 16], included UML class diagrams [5, 2, 6]. This allows for exploiting DL reasoning services for implementing various forms of reasoning on such diagrams.

In this paper we discuss encodings of UML class diagrams into two DLs, namely $\mathcal{DLR}_{ifd}$ [8, 7], i.e., $\mathcal{DLR}$ with identification constraints and functional dependencies, and $\mathcal{ALCQI}$. The semantics of $\mathcal{DLR}_{ifd}$ *fully captures* UML class diagrams, while $\mathcal{ALCQI}$ is the DL directly supported by state-of-the-art DL reasoning systems [22, 17], and thus facilitates the use of such systems as the core engines for advanced UML CASE tools.

We experimented such encodings by exploiting the reasoning services offered by state-of-the-art automated reasoning systems, within an ongoing project in collaboration with IBM Tivoli, obtaining very encouraging results on industrial relevant diagrams.

The rest of the paper is organized as follows. In Section 2 we give an overview of UML class diagrams and for each construct we provide a $\mathcal{DLR}_{ifd}$-based formalization. In Section 3 we discuss a formalization of UML class diagrams based on $\mathcal{ALCQI}$ and show how and why we can pass from the first to the second one. In Section 4 we present our experiments and discuss them. Section 5 concludes the paper.

2 Representing UML class diagrams

In this section, we briefly illustrate UML class diagrams and for each construct, we provide an encoding in terms of $\mathcal{DLR}_{ifd}$. We refer the reader to [8, 7] for the syntax and semantics of this DL. Scope of this section is to show that $\mathcal{DLR}_{ifd}$ *fully captures* the semantics of UML class diagrams for the conceptual perspective [15].

Classes A *class* in an UML class diagram denotes a *set of objects* with common features, hence it can be represented in $\mathcal{DLR}_{ifd}$ by a concept, since also $\mathcal{DLR}_{ifd}$ concepts denote *sets of objects*.

An *attribute* a of type[1] T for a class C associates to each instance of C a set of instances of T[2]. It can be captured in $\mathcal{DLR}_{ifd}$ by means of a binary relation a between instances of C and instances of T and by the following assertion:

$$C \sqsubseteq \forall[1](a \Rightarrow (2 : T)),$$

stating that each relation a having C as first component, has T as second component. An optional multiplicity $[i..j]$ for a specifies that a associates to each instance of C at least i and most j instances of T. When the multiplicity is missing, [1..1] is assumed. Multiplicity can be naturally captured by number restrictions, as follows:

$$C \sqsubseteq (\geq i\,[1]a) \sqcap (\leq j\,[1]a).$$

An *operation* $f(P_1, \ldots, P_m) : (R_1, \ldots, R_n)$ of a class C is a *function* that associates to an m-tuple of parameters belonging to the classes $P_1, \ldots, P_m$, respectively, an n-tuple of return values belonging to the classes $R_1, \ldots, R_n$, respectively. Recalling that $\mathcal{DLR}_{ifd}$ has the ability of representing n-ary relations

[1] For simplicity, we consider types as classes, without distinguishing between classes and domains, such as integers, reals, ...

[2] Note that UML allows one to define two classes having the same attribute, possibly of different types.

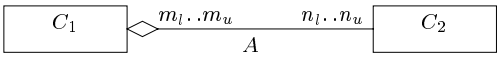


Figure 1: Aggregation in UML

and functional dependencies on them, we capture the operation above by means of a relation named $\mathtt{op}_{f(P_1,\ldots,P_m):(R_1,\ldots,R_n)}$ of arity $m+n+1$, among instances of the concepts $C, P_1, \ldots, P_m, R_1, \ldots, R_n$. On such a relation we impose:

- that parameters and return values have correct types:

$$C \sqsubseteq \forall[1](\mathtt{op}_{f(P_1,\ldots,P_m):(R_1,\ldots,R_n)} \Rightarrow ((2:P_1) \sqcap \cdots \sqcap (m+1:P_m) \sqcap (m+2:R_1) \sqcap \cdots \sqcap (m+n+1:R_n)));$$

- functional dependencies from the class the operation belongs to and the input parameters, to each return value:

$$\begin{gathered}(\textbf{fd } \mathtt{op}_{f(P_1,\ldots,P_m):(R_1,\ldots,R_n)}\ 1,\ldots,m+1 \rightarrow m+2),\\ \ldots,\\ (\textbf{fd } \mathtt{op}_{f(P_1,\ldots,P_m):(R_1,\ldots,R_n)}\ 1,\ldots,m+1 \rightarrow m+n+1),\end{gathered}$$

  such assertions state that each return value is uniquely determined by the object of invocation and the input parameters.

Observe that the formalization of operations in $\mathcal{DLR}_{ifd}$ allows one to have operations with the same name or even with the same signature in two different classes. In particular, following UML specifications, our framework allows for *overloading* of methods, that takes place between two or more functions having the same name but different signatures. *Overriding* requires that two methods have the same name and signature, but behave in different ways. UML class diagrams for the conceptual perspective do not specify bodies of methods but only their signature, hence overriding becomes immaterial and so does in our representation.

Aggregations An *aggregation* in UML is a binary relation between the instances of two classes, denoting a generic form of part-whole relationship: aggregation A in Figure 1 indicates that the instances of the (containing) class C_1 have components that are instances of the (contained) class C_2. This is formalized by means of a binary relation A together with the assertion

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

The distinction between the contained class and the containing class is not lost, since we can use the convention that the first argument of the relation is the containing class.

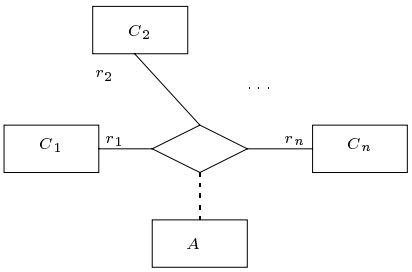


Figure 2: Association in UML

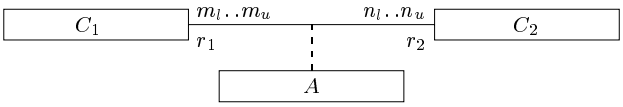


Figure 3: Binary association in UML

The multiplicity of an aggregation can be easily expressed. For example, the multiplicities shown in Figure 1 are formalized in $\mathcal{DLR}_{ifd}$ as:

$$\begin{aligned} C_1 &\sqsubseteq (\geq n_l\,[1]A) \sqcap (\leq n_u\,[1]A), \\ C_2 &\sqsubseteq (\geq m_l\,[2]A) \sqcap (\leq m_u\,[2]A). \end{aligned}$$

Associations An *association* in UML, graphically rendered as in Figure 2, 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.

We model in $\mathcal{DLR}_{ifd}$ an n-ary association A between classes $C_1, \ldots C_n$, that has not a related association class, by means of an n-ary relation A, and the assertion

$$A \sqsubseteq (1 : C_1) \sqcap \ldots \sqcap (n : C_n),$$

stating that A has C_i as i-th component, for $i = 1, \ldots, n$.

If A has a related association class, we formalize it in a reified way: we introduce a concept A and n *binary* relations $r_1, \ldots, r_n$, one for each component of the association A. Each binary relation r_i has A as its first component and as C_i its second component. Then we introduce the following assertions:

$$\begin{aligned} A \;\sqsubseteq\; & \exists[1]r_1 \sqcap (\leq 1\,[1]r_1) \sqcap \forall[1](r_1 \Rightarrow (2 : C_1)) \sqcap \;\ldots \sqcap \\ & \exists[1]r_n \sqcap (\leq 1\,[1]r_n) \sqcap \forall[1](r_n \Rightarrow (2 : C_n)) \end{aligned}$$

$$(\textbf{id}\ A\ [1]r_1, \ldots, [1]r_n).$$

The first assertion states that the concept A must have all components $r_1, \ldots, r_n$ of the association A and that each such component is single-valued; it also specifies the class each component has to belong to. The second assertion specifies

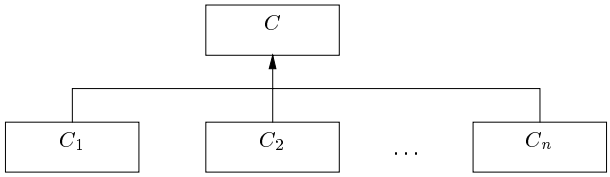


Figure 4: A class hierarchy in UML

that each instance of A represents a *distinct* tuple in $C_1 \times \cdots \times C_n$. We remind that an important feature of $\mathcal{DLR}_{ifd}$ is the ability of capturing identification constraints on concepts.

For a binary UML association[3], we can easily represent multiplicities by imposing suitable number restrictions on the relations modeling the components of the association. The multiplicities shown in Figure 3 are captured by:

$$\begin{array}{lcl} C_1 & \sqsubseteq & (\geq n_l\,[2](r_1 \sqcap (1:A))) \sqcap (\leq n_u\,[2](r_1 \sqcap (1:A))) \\ C_2 & \sqsubseteq & (\geq m_l\,[2](r_2 \sqcap (1:A))) \sqcap (\leq m_u\,[2](r_2 \sqcap (1:A))). \end{array}$$

Generalization In UML, a *generalization* between a parent class and a child class, as shown in Figure 4, is used to specify that each instance of the child class is also an instance of the parent class. The instances of the child class inherit the properties of the parent class, and usually they satisfy additional properties that do not hold for the parent class.

Generalization is naturally supported in $\mathcal{DLR}_{ifd}$. We can express that an UML class C generalizes n classes $C_1, \ldots, C_n$, by the set of assertions

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

Inheritance between $\mathcal{DLR}_{ifd}$ concepts works exactly as inheritance between UML classes, as a consequence of the semantics of inclusion assertions, which is based on sub-setting: 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. Hence, each attribute or operation of C_2, and each aggregation and association involving C_2 is correctly inherited by C_1. The above formalization 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 impose covering or mutual disjointness between classes. If the superclass C is a *covering* of the subclasses $C_1, \ldots, C_n$, we include the additional assertion

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

[3]In UML multiplicities are look-across cardinality constraints [24]. This makes their use in non-binary associations difficult w.r.t. both modeling and reasoning.

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

$$C_i \sqsubseteq \neg C_j.$$

Constraints In UML it is possible to add *constraints*, in order to express application semantics which cannot be expressed by other constructs of UML class diagrams. One can exploit the expressive power of $\mathcal{DLR}_{ifd}$ to formalize several types of constraints (although not all those that OCL [23], i.e. first order logic, can), that can be taken fully into account when reasoning on class diagrams.

3 Representing UML class diagrams in $\mathcal{ALCQI}$

$\mathcal{DLR}_{ifd}$ *fully* captures the semantics of UML class diagrams (except for general OCL constructs). Functional dependencies and identification constraints play a special role since allow one to impose that each instance of a concept representing an n-ary association A (with a related association class) is univocally identified by each first component of the n roles r_i and, similarly, that each return value of an operation f of arity m belonging to class C is determined once the instances of the class and of the input parameters are given. Current state-of-the-art DL-based reasoning systems do not support functional dependencies and identification constraints yet: these require advanced forms of reasoning on generalized ABoxes that are not implemented yet [8]. In [8] it is shown that for a KB without ABox, logical implication of inclusion assertions (note, not of functional dependency and identification constraint assertions) can be verified without considering functional dependencies and identification constraints at all. This is because a $\mathcal{DLR}_{ifd}$ TBox has the tree model property [7], i.e., if a TBox is satisfiable, it admits a model having a tree structure, and since in such models no tuples exist having more than one component in common, identification constraints and functional dependencies are trivially satisfied. We are exactly in this setting. Hence, if we limit to ask queries not involving either functional dependencies or identification constraints, we can drop them, obtaining a KB in a DL named $\mathcal{DLR}$. $\mathcal{DLR}$ KBs can be translated into $\mathcal{ALCQI}$ KBs[4] and we can exploit DL reasoning systems to deduce relevant properties of UML class diagrams. However, this brings about a disruptive mixture of inverse roles with terminological cycles involving existentials, and with functional restrictions combined with existential restrictions. In fact, in [2], we found out that state-of-the-art reasoners have difficulties in classifying KBs obtained in such a way[5].

[4] Actually, in the variants of $\mathcal{ALCQI}$ accepted by the reasoners.

[5] In fact, techniques that are currently being developed to deal with complex KBs (see [21]) can be of help.

To overcome this problem, we propose a direct encoding of UML class diagrams in $\mathcal{ALCQI}$, in order to obtain KBs on which reasoning services are easier for the reasoning systems. In particular, the encoding of UML class diagram constructs is as follows:

- *classes* are modeled by $\mathcal{ALCQI}$ concepts; *attribute* a of type T for class C is captured by a binary relation a and by the assertion $C \sqsubseteq (\geq i\, a) \sqcap (\leq j\, a)$; if the attribute has associated a multiplicity $[i..j]$, this is captured by $C \sqsubseteq (\geq i\, a) \sqcap (\leq j\, a)$; *operation* $f(P_1, \ldots, P_m) : (R_1, \ldots, R_n)$ of a class C is represented in a reified way, i.e., by introducing an atomic concept $\mathtt{op}_{f(P_1,\ldots,P_m):(R_1,\ldots,R_n)}$, $n + m + 1$ roles $r_1, \ldots, r_{n+m+1}$ and the following assertion: $\mathtt{op}_{f(P_1,\ldots,P_m):(R_1,\ldots,R_n)} \sqsubseteq \forall r_1.C \sqcap \exists r_1 \sqcap (\leq 1\, r_1) \sqcap \forall r_2.P_1 \sqcap \exists r_2 \sqcap (\leq 1\, r_2) \sqcap \cdots \sqcap \forall r_{m+n+1}.R_n \sqcap \exists r_{n+m+1} \sqcap (\leq 1\, r_{m+n+1})$;

- *aggregation* A in Figure 1 between the containing class C_1 and the contained class C_2 is modeled by $\top \sqsubseteq \forall A.C_2 \sqcap \forall A^-.C_1$; the multiplicity constraints shown there are captured by $C_1 \sqsubseteq (\geq n_l\, A) \sqcap (\leq n_u\, A)$ and $C_2 \sqsubseteq (\geq m_l\, A^-) \sqcap (\leq m_u\, A^-)$;

- *n-ary association* A between classes $C_1, \ldots C_n$, shown in Figure 2, is captured by reifying the association and by the assertion $A \sqsubseteq (\leq 1\, r_1) \sqcap \cdots \sqcap (\leq 1\, r_n) \sqcap \exists r_1.C_1 \sqcap \cdots \sqcap \exists r_n.C_n$, both if it has and it has not a related association class; the multiplicity constraints shown in Figure 3 are represented by $C_1 \sqsubseteq (\geq n_l\, r_1^-.A) \sqcap (\leq n_u\, r_1^-.A)$ and $C_2 \sqsubseteq (\geq m_l\, r_2^-.A) \sqcap (\leq m_u\, r_2^-.A)$.

It can be shown that such encoding is equivalent to the $\mathcal{DLR}$ one. In particular, the ability of correctly representing an n-ary relation through reification, i.e. with a concept plus one role for each relation component, is again granted by the tree model property.

4 Experimentation

We performed several experiments on UML class diagrams. In this section we report results on (successful) classification of diagrams, available on the site `http://www.dis.uniroma1.it/~berardi/uml2dl`. The Table in Figure 5 gives an idea of how easier are the KBs obtained from the encoding above, for the reasoners. In particular, we have used the two systems FaCT [19, 22, 20][6] (the executable $\mathcal{SHIQ}$ reasoner (shiq-app.exe) contained in the CORBA-FaCT distribution v.3.1, excluding the CORBA interface) and Racer [18, 17][7] (v.1-6-7). We have run all our experiments on a Pentium III biprocessor, 866 Mhz, 512MB of RAM and OS Windows 2000 Professional. The first set of columns

[6] Available at `http://www.cs.man.ac.uk/~horrocks/FaCT`.

[7] Available at `http://kogs-www.informatik.uni-hamburg.de/~race`.

UML class diagrams	$\mathcal{DLR}$-based encoding		$\mathcal{ALCQI}$-based encoding	
	FaCT	Racer	FaCT	Racer
Restaurant	yes	no	yes	yes
Soccer	no	no	yes	yes
Library	yes	no	yes	yes
Hospital	yes	no	yes	yes
School-exam	no	no	yes	yes
Video library	no	no	yes	yes
Workshop	no	no	yes	yes

Figure 5: Successful classification of the presented KBs.

in the table reports the results of the classification of the KBs obtained from a $\mathcal{DLR}$-based encoding of UML class diagrams and the second one reports the results of the classification of the KBs obtained following the $\mathcal{ALCQI}$-based encoding. Besides, "yes" means that the reasoners can classify the KBs, "no" that they cannot, because they run out of resources. As one can see, both reasoners can classify all the presented KBs obtained with the $\mathcal{ALCQI}$-based encoding, whereas this does not happen with the $\mathcal{DLR}$-based encoding.

We also performed several experiments on UML class diagrams as part of an ongoing project in collaboration with IBM Tivoli. Scope of this project is to study how to extract intensional knowledge from Common Information Model, that can be useful to support the management of an elaboration system. In particular, we want to exploit reasoning systems to derive such knowledge, from an arbitrarily complex UML class diagram.

Common Information Model (CIM) is a model based on UML with the purpose of providing a rigorous approach for modelling systems and networks using the object-oriented paradigm. CIM has a *Meta Schema* that specifies how other schemas can be constructed, in order to form the basis for a sort of vocabulary for analyzing and describing managed systems. CIM offers two main conceptual layers, that form the *CIM Schema*: the *Core Model*, that captures basic notions, common to all areas of management, and the *Common Model*, that expresses concepts related to specific management areas (e.g., device, networks, systems, etc.). In our experiments, we focused on subschemas of CIM Schema v.26[8].

In our experiments we considered both the $\mathcal{DLR}$-based encoding and the $\mathcal{ALCQI}$-based one. FaCT and Racer can classify *all* the KBs presented in a few seconds. This is due to the fact that they have few multiplicity constraints on associations and aggregations, which leads to few terminological cycles. As we expected, with the latter encoding, reasoners take less than with the former one, since the KBs obtained are much simpler.

[8]They are available on the page http://www.dmtf.org/standards/cim_schema_v26.php.

5 Conclusions

In this paper, we have discussed two encodings of UML class diagrams in terms of $\mathcal{DLR}_{ifd}$ and of $\mathcal{ALCQI}$. The $\mathcal{DLR}_{ifd}$ encoding allows us to capture exactly the semantics of UML class diagram constructs, since $\mathcal{DLR}_{ifd}$ can represent n-ary relations, functional dependencies on relations and identification constraints on concepts. By dropping functional dependencies and identification constraints, that cannot be dealt with yet, one can use the state-of-the-art reasoners. However, we noticed that they have still difficulties in reasoning on KBs exploiting the $\mathcal{DLR}$-based encoding of complex UML class diagrams. Hence we have devised a direct encoding into $\mathcal{ALCQI}$, which is the DL adopted by state-of-the-art DL-based reasoning systems. The experiments we have reported show that the KBs obtained in this case are easily classified by the reasoners. The use of $\mathcal{DLR}_{ifd}$ (but also of $\mathcal{DLR}$) remains a challenge for the future DL-based systems.

Acknowledgments. The author would like to thank Giuseppe De Giacomo and Diego Calvanese for their support during the writing of this paper.

References

[1] F. Baader, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2002. To appear.

[2] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams using description logic based systems. In *Proc. of the KI'2001 Workshop on Applications of Description Logics*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-44/`, 2001.

[3] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Trans. on Database Systems*, 17(3):385–422, 1992.

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

[5] 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.

[6] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. A formal framework for reasoning on uml class diagrams. In *Proc. of the 13th Int. Sym. on Methodologies for Intelligent Systems (ISMIS 2002)*, pages 503–513, 2002.

[7] 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.

[8] D. Calvanese, G. De Giacomo, and M. Lenzerini. Identification constraints and functional dependencies in description logics. In *Proc. of IJCAI 2001*, pages 155–160, 2001.

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

[10] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *J. of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

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

[12] A. Evans, R. France, K. Lano, and B. Rumpe. The UML as a formal modeling notation. In *Proc. of the OOPSLA'97 Workshop on Object-oriented Behavioral Semantics*, pages 75–81, 1997.

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

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

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

[16] E. Franconi and G. Ng. The i.com tool for intelligent conceptual modeling. In *Proc. of KRDB 2000*, pages 45–53. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-29/`, 2000.

[17] V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of IJCAI 2001*, pages 161–168, 2001.

[18] V. Haarslev and R. Möller. RACER system description. In *Proc. of IJCAR 2001*, volume 2083 of *LNAI*, pages 701–705. Springer-Verlag, 2001.

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

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

[21] I. Horrocks and U. Sattler. Optimised reasoning for $\mathcal{SHIQ}$. In *Proc. of the 15th Eur. Conf. on Artificial Intelligence (ECAI 2002)*, July 2002.

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

[23] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley Publ. Co., Reading, Massachussetts, 1998.

[24] B. Thalheim. Fundamentals of cardinality constraints. In *Proc. of ER'92*, pages 7–23. Springer-Verlag, 1992.

An Approach for Optimized Approximation

Sebastian Brandt and Anni-Yasmin Turhan*

Theoretical Computer Science, TU Dresden, Germany
Email: {brandt, turhan}@tcs.inf.tu-dresden.de

Abstract

Approximation is a new inference service investigated in [4]. An approximation of an $\mathcal{ALC}$-concept by an $\mathcal{ALE}$-concept can be computed in double exponential time. Consequently, one needs powerful optimization techniques for approximating an entire unfoldable TBox. Addressing this issue we identify a special form of $\mathcal{ALC}$-concepts that can be divided into parts s.t. each part can be approximated independently.

1 Motivation

This paper presents preliminary results on optimization techniques for the computation of approximations. Approximation is a new non-standard inference service in Description Logics (DLs) introduced in [4]. Approximating a concept, defined in one DL, means to translate this concept to another concept, defined in a second, typically less expressive DL, such that both concepts are as closely related as possible with respect to subsumption. Like other non-standard inferences such as the least common subsumer (lcs) or matching, approximation has been introduced to support the construction and maintenance of DL knowledgebases (see [9, 5]). Approximation has a number of different applications some of which we will mention here, see [4] for others.

Computation of commonalities of concepts. Given a set of concepts the problem is to extract the commonalities of the input concepts. Typically, the lcs is employed for this task. In case a DL $\mathcal{L}$ provides concept disjunction, the lcs is just the disjunction of C_1 and C_2 ($C_1 \sqcup C_2$). Thus, a user inspecting this concept does not learn anything about the commonalities between C_1 and C_2. By using approximation, however, one can make the commonalities explicit to some extent by first approximating C_1 and C_2 in a sublanguage of $\mathcal{L}$ which does not provide disjunction, and then computing the lcs of the approximations in $\mathcal{L}$.

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

Translation of knowledge-bases. Approximation can be used to (automatically) translate a knowledge-base written in an expressive DL into another (semantically closely related) knowledge-base in a less expressive DL. The translation may become necessary to port knowledge-bases between different knowledge representation systems or to integrate different knowledge-bases.

We investigate the case of translating an $\mathcal{ALC}$-TBox into an $\mathcal{ALE}$-TBox by computing the approximation of each concept defined in the $\mathcal{ALC}$-TBox. In [4], a first in-depth investigation of the approximation inference has been presented. In particular, a double-exponential time algorithm has been devised to approximate $\mathcal{ALC}$-concepts by $\mathcal{ALE}$-concepts. Consequently, approximating an entire TBox requires substantial optimizations. We address this problem by identifying a form of $\mathcal{ALC}$-concept descriptions whose conjuncts can be approximated independently. This approach speeds-up the computation of a single approximation. Moreover, it also allows to re-use an obtained approximation in subsequent approximations by simply inserting the approximation of a subconcept in the current approximation. Therefore the splitting of concepts in independent parts is a prerequisite for applying caching techniques to approximation. The full proofs of the results presented here can be found in our technical report [6].

2 Preliminaries

Concept descriptions are inductively defined based on a set of concept *constructors* starting with a set N_C of *concept names* and a set N_R of *role names*. In this paper, we consider concept descriptions built from the constructors shown in Table 1 where C and D denote arbitrary concepts, A a concept name, and r a role. Note that in $\mathcal{ALC}$ every concept description can be negated whereas in $\mathcal{ALE}$ negation is only allowed in front of concept names. In the following a concept description formed with the constructors allowed in a DL $\mathcal{L}$ is called *$\mathcal{L}$-concept description*.

As usual, the semantics of a concept description is defined in terms of an *interpretation* $\mathcal{I} = (\Delta, \cdot^I)$. The domain Δ of $\mathcal{I}$ is a non-empty set and the interpretation function $\cdot^I$ maps each concept name $A \in N_C$ to a set $A^I \subseteq \Delta$ and each role name $r \in N_R$ to a binary relation $r^I \subseteq \Delta \times \Delta$. The extension of $\cdot^I$ to arbitrary concept descriptions is defined inductively, as shown in Table 1.

For the sake of simplicity, we assume that the set N_R of role names is the singleton $\{r\}$. However, all definitions and results can easily be generalized to arbitrary sets of role names. We also assume that each conjunction in an $\mathcal{ALE}$-concept description contains at most one value restriction of the form $\forall r.C'$ (this is w.l.o.g. due to the equivalence $\forall r.E \sqcap \forall r.F \equiv \forall r.(E \sqcap F)$).

A *TBox* is a finite set of concept definitions of the form $A \doteq C$, where $A \in N_C$ and C is a concept description. In addition, we require that TBoxes

Syntax	Semantics	$\mathcal{ALE}$	$\mathcal{ALC}$
$\top$	Δ	x	x
$\bot$	$\emptyset$	x	x
$C \sqcap D$	$C^I \cap D^I$	x	x
$\exists r.C$	$\{x \in \Delta \mid \exists y : (x,y) \in r^I \wedge y \in C^I\}$	x	x
$\forall r.C$	$\{x \in \Delta \mid \forall y : (x,y) \in r^I \rightarrow y \in C^I\}$	x	x
$\neg A$, $A \in N_C$	$\Delta \setminus A^I$	x	x
$\neg C$	$\Delta \setminus C^I$		x
$C \sqcup D$	$C^I \cup D^I$		x

Table 1: Syntax and semantics of concept descriptions.

are *unfoldable*, i.e., they are acyclic and do not contain multiple definitions (see, e.g., [10]). Concept names occurring on the left-hand side of a definition are called *defined concepts*. All other concept names are called *primitive concepts*. In TBoxes of the DL $\mathcal{ALE}$, negation may only be applied to primitive concepts. An interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$ iff it satisfies all its concept definitions, i.e., $A^{\mathcal{I}} = C^{\mathcal{I}}$ for all definitions $A \doteq C$ in $\mathcal{T}$.

One of the most important traditional inference services provided by DL systems is computing the subsumption hierarchy. The concept description C is *subsumed* by the description D ($C \sqsubseteq D$) iff $C^I \subseteq D^I$ holds for all interpretations $\mathcal{I}$; C and D are *equivalent* ($C \equiv D$) iff $C \sqsubseteq D$ and $D \sqsubseteq C$. Subsumption and equivalence in $\mathcal{ALC}$ is PSPACE-complete [11] and NP-complete in $\mathcal{ALE}$ [7].

2.1 $\mathcal{ALE}$-Approximation for $\mathcal{ALC}$

In order to approximate $\mathcal{ALC}$-concept descriptions by $\mathcal{ALE}$-concept descriptions, we need to compute the lcs in $\mathcal{ALE}$.

Definition 1 *Given $\mathcal{L}$-concept descriptions $C_1, \ldots, C_n$ with $n \geq 2$ for some description logic $\mathcal{L}$, the $\mathcal{L}$-concept description C is the* least common subsumer *(lcs) of $C_1, \ldots, C_n$ ($C = \mathsf{lcs}(C_1, \ldots, C_n)$ for short) iff (i) $C_i \sqsubseteq C$ for all $1 \leq i \leq n$, and (ii) C is the least concept description with this property, i.e., if C' satisfies $C_i \sqsubseteq C'$ for all $1 \leq i \leq n$, then $C \sqsubseteq C'$.*

As already mentioned, in $\mathcal{ALC}$ the lcs trivially exists since $lcs(C, D) \equiv C \sqcup D$. For $\mathcal{ALE}$ the existence is not obvious. It was shown in [2] that the lcs of two or more $\mathcal{ALE}$-concept descriptions always exists, that its size may grow exponentially in the size of the input descriptions, and that it can be computed in exponential time.

Intuitively, to approximate an $\mathcal{ALC}$-concept description from "above" means to compute an $\mathcal{ALE}$-concept description that is more general than the input concept description but minimal w.r.t. subsumption.

Definition 2 *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two DLs, and let C be an $\mathcal{L}_1$- and D be an $\mathcal{L}_2$-concept description. Then, D is called an* upper $\mathcal{L}_2$-approximation *of C ($D = approx_{\mathcal{L}_2}(C)$ for short) iff (i) $C \sqsubseteq D$, and (ii) D is minimal with this property, i.e., $C \sqsubseteq D'$ and $D' \sqsubseteq D$ implies $D' \equiv D$ for all $\mathcal{L}_2$-concept descriptions D'.*

Although defined in [4] lower approximations are not yet further investigated. In this paper, we restrict our investigations to upper $\mathcal{ALE}$-approximations of $\mathcal{ALC}$-concept descriptions. Therefore, whenever we speak of approximations, we mean upper $\mathcal{ALE}$-approximations. Thus, having defined approximation we turn now to how to actually compute them.

2.2 The Approximation Algorithm

Before a defined concept from a TBox can be approximated it has to be *unfolded* w.r.t. the underlying TBox to make the information captured in the concept definitions explicit. To this end, every defined concept is replaced by the concept description on the right-hand side of its concept definition until no defined concept occurs in the concept description. It is well known that this process can cause an exponential blow-up of the concept description, see [10]. To recapitulate the approximation algorithm presented in [4], we need to introduce the $\mathcal{ALC}$-normal form.

For an unfolded concept description C the *role-depth* $rd(C)$ is inductively defined as follows:

$$\begin{aligned} rd(N) &:= 0 && \text{, where } N \in N_C \cup \{\bot, \top\} \\ rd(\neg C) &:= rd(C) \\ rd(C_1 \ \rho \ C_2) &:= \max\{rd(C_1), rd(C_2)\} && \text{, where } \rho \in \{\sqcap, \sqcup\} \\ rd(Qr.C) &:= 1 + rd(C) && \text{, where } Q \in \{\exists, \forall\} \end{aligned}$$

A *role-level* of a concept C is the set of all concept descriptions occurring on the same role-depth in C. The topmost role-level of a concept description is called its *top-level*.

We call a concept description *top-level $\sqcup$-free* if it is in negation normal form (NNF), i.e., negation is pushed inwards until in front of a concept name, and does not contain any disjunction on top-level. Some notation is needed to access the different parts of an $\mathcal{ALE}$-concept description or a top-level $\sqcup$-free $\mathcal{ALC}$-concept description C:

- $\mathsf{prim}(C)$ denotes the set of all (negated) concept names and the bottom concept occurring on the top-level of C;
- $\mathsf{val_r}(C) := C_1 \sqcap \cdots \sqcap C_n$, if there exist value restrictions of the form $\forall r.C_1, \ldots, \forall r.C_n$ on the top-level of C; otherwise, $\mathsf{val_r}(C) := \top$;

Input: unfolded $\mathcal{ALC}$-concept description C
Output: $\mathcal{ALE}$-approximation of C

1. If $C \equiv \bot$, then $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C) := \bot$;
   if $C \equiv \top$, then $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C) := \top$
2. Otherwise, transform C into $\mathcal{ALC}$-normal form $C_1 \sqcup \cdots \sqcup C_n$ and return $\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C) :=$

$$\bigsqcap_{A \in \bigcap_{i=1}^{n} \mathsf{prim}(C_i)} A \quad \sqcap \quad \forall r.\mathsf{lcs}\{\mathsf{c\text{-}approx}_{\mathcal{ALE}}(\mathsf{val}_\mathsf{r}(C_i)) \mid 1 \le i \le n\} \ \sqcap$$
$$\bigsqcap_{(C_1',\ldots,C_n') \in \mathsf{ex}_\mathsf{r}(C_1) \times \cdots \times \mathsf{ex}_\mathsf{r}(C_n)} \exists r.\mathsf{lcs}\{\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C_i' \sqcap \mathsf{val}_\mathsf{r}(C_i)) \mid 1 \le i \le n\}$$

Figure 1: The recursive algorithm $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$.

- $\mathsf{ex}_\mathsf{r}(C) := \{C' \mid \text{there exists } \exists r.C' \text{ on the top-level of } C\}$.

Equipped with these we can define the $\mathcal{ALC}$-normal form in which conjuncts are distributed over the disjuncts. An arbitrary $\mathcal{ALC}$-concept description is transformed into a concept description with at most one disjunction on top-level of every concept of each role-level.

Definition 3 *An $\mathcal{ALC}$-concept description C is in $\mathcal{ALC}$-*normal form *iff*

1. *if $C \equiv \bot$, then $C = \bot$; if $C \equiv \top$, then $C = \top$;*
2. *otherwise, C is of the form $C = C_1 \sqcup \cdots \sqcup C_n$ with*

$$C_i = \bigsqcap_{A \in \mathsf{prim}(C_i)} A \sqcap \bigsqcap_{C' \in \mathsf{ex}_\mathsf{r}(C_i)} \exists r.C' \sqcap \forall r.\mathsf{val}_\mathsf{r}(C_i),$$

$C_i \not\equiv \bot$, and $\mathsf{val}_\mathsf{r}(C_i)$ and every concept description in $\mathsf{ex}_\mathsf{r}(C_i)$ is in $\mathcal{ALC}$-normal form, for all $i = 1, \ldots, n$.

Obviously, every $\mathcal{ALC}$-concept description can be turned into an equivalent concept description in $\mathcal{ALC}$-normal form. Every disjunct of a concept in $\mathcal{ALC}$-normal form is top-level $\sqcup$-free. Unfortunately, the normalization may take exponential time. For instance, the normal form of $(A_1 \sqcup A_2) \sqcap \cdots \sqcap (A_{2n-1} \sqcup A_{2n})$ is of size exponential in n.

The approximation algorithm displayed in Figure 1 checks if the input is a concept equivalent to $\top$ or $\bot$—in this case the approximation is trivial—otherwise it proceeds recursively on the $\mathcal{ALC}$-normal form of the input and extracts the commonalities of all disjuncts. Unfortunately, the algorithm needs double-exponential time for arbitrary $\mathcal{ALC}$-concepts in the worst case. Despite

its high complexity, our prototypical implementation of the algorithm showed a quite promising performance in respect to run-time and resulting concept sizes, for details see [4].

3 Optimizing $\mathcal{ALE}$-Approximations

A TBox can be translated by computing the approximation of the concept description on the right-hand side of every concept definition in the TBox. Each defined concept has to be unfolded and transformed into $\mathcal{ALC}$-normal form before the approximation algorithm can be applied. Unfortunately, both of these steps cause an exponential growth of the concept description.

For standard reasoning tasks [1, 8] and also for the computation of the lcs [3] the first source of complexity can often be alleviated by *lazy unfolding*. Here the idea is to replace a defined concept in a concept description only if examination of that part of the description is necessary. Lazy unfolding unfolds all defined concepts appearing on the top-level of the concept description under consideration while defined concepts on deeper role-levels remain unchanged as long as possible.

When computing the lcs the main benefit of lazy unfolding is that in some cases defined concepts can be used directly in the lcs concept description. If, for example a defined concept C appears in all input concept descriptions on the same role-level, the concept definition of C does not need to be processed, but C can be inserted into the lcs directly, see [3] for details. In the case of approximation, however, this effect of lazy unfolding can not be utilized even if a defined concept is obviously common to all disjuncts. For example, in $(A \sqcap C) \sqcup (C \sqcap (\neg B))$ the concept name C cannot be used directly as a name in the approximation because the $\mathcal{ALC}$-concept description C stands for must be approximated. Thus unfolding a concept completely cannot be avoided for approximation.

The double-exponential time complexity of the approximation algorithm, however, suggests another approach to optimization. Instead of approximating an input concept C as a whole a significant amount of time could be saved by splitting C into its conjuncts and approximating them separately. If, for instance, C consists of two conjuncts of size n then the approximation of C takes some $a^{b^{2n}}$ steps while the conjunct-wise approach would just take $2a^{b^n}$. Unfortunately, splitting an arbitrary input concept at conjunctions leads to incorrect approximations, as examples show [4]. In the following section we will therefore introduce a class of so-called nice $\mathcal{ALC}$-concepts for which the conjunct-wise approximation still produces the correct result.

3.1 Nice Concepts

In the following we assume that all concept descriptions are unfolded and in NNF. For an $\mathcal{ALC}$-concept description C and $i \in \mathbb{N}$ the *quantor set* $Q_r(C,i)$ denotes the set of quantors used on the role-level i of C (referring to role r). Hence, for $0 \leq i \leq rd(C)$ the quantor set $Q_r(C,i)$ is a nonempty subset of $\{\forall, \exists\}$. Similarly, the *name set* $N_r(C,i)$ denotes the atomic concepts used on a specific role-level. Formally, Q and N are defined as follows.

Definition 4 *Let $C := \sqcup_{i=1}^{k} C_i$ be an $\mathcal{ALC}$-concept description in $\mathcal{ALC}$-normal form. For $d \in \mathbb{N}$, the sets $Q_r(C,d)$ and $N_r(C,d)$ are inductively defined by:*

- $Q_r(C,0) := \{\exists \mid \bigcup_{i=1}^{k} \mathsf{ex_r}(C_i) \neq \emptyset\} \cup \{\forall \mid \sqcap_{i=1}^{k} \mathsf{val_r}(C_i) \not\equiv \top\}$

  $N_r(C,0) := \bigcup_{i=1}^{k} \mathsf{prim}(C_i)$

- $Q_r(C,d+1) := \bigcup_{i=1}^{k} \bigcup_{C' \in \mathsf{ex_r}(C_i)} Q_r(C',d) \cup \bigcup_{i=1}^{k} Q_r(\mathsf{val_r}(C_i),d)$

  $N_r(C,d+1) := \bigcup_{i=1}^{k} \bigcup_{C' \in \mathsf{ex_r}(C_i)} N_r(C',d) \cup \bigcup_{i=1}^{k} N_r(\mathsf{val_r}(C_i),d).$

For a concept description C not in $\mathcal{ALC}$-normal form, Q and N are defined in terms of the $\mathcal{ALC}$-normal form of C. For example the unfolded concept $C = (\exists r.(A \sqcap B) \sqcap \forall r.(D \sqcup (\exists r.\neg E)))$ has the quantor sets $Q_r(C,0) = \{\forall, \exists\}$, $Q_r(C,1) = \{\exists\}$ and $Q_r(C,i) = \emptyset$ for $i \geq 2$. For the name sets, we have $N(C,0) = \emptyset$, $N(C,1) = \{A, B, D\}$, and $N(C,2) = \{\neg E\}$.

We are now ready to specify in detail what nice concepts are. In general, an approximation $approx_{\mathcal{ALE}}(C \sqcap D)$ cannot be split at the conjunction because of possible interactions between existential and value restrictions on the one hand and inconsistencies induced by negation on the other. For example, the approximation $approx_{\mathcal{ALE}}(\exists r.\top \sqcap (\forall r.A \sqcup \exists r.A))$ yields $\exists r.A$ while the split version $approx_{\mathcal{ALE}}(\exists r.\top) \sqcap approx_{\mathcal{ALE}}(\forall r.A \sqcup \exists r.A)$ only produces $\exists r.\top$. Similarly, the conjunction $A \sqcap (\neg A \sqcup B)$ cannot be approximated separately.

We now call those concepts *nice* for which this simplified strategy still produces the correct result and for which a simple syntactic discrimination rule exists. Firstly, the role quantors occurring in nice concepts are restricted to one type per role-level. Hence, on every role-level of a nice concept either no $\forall$-restrictions or no $\exists$-restrictions occur. Secondly, a concept name and its negation may not occur on the same role-level. Consider Figure 2 for an illustration of these rules. Formally, we can define nice concepts by means of the syntactical operators from Definition 4.

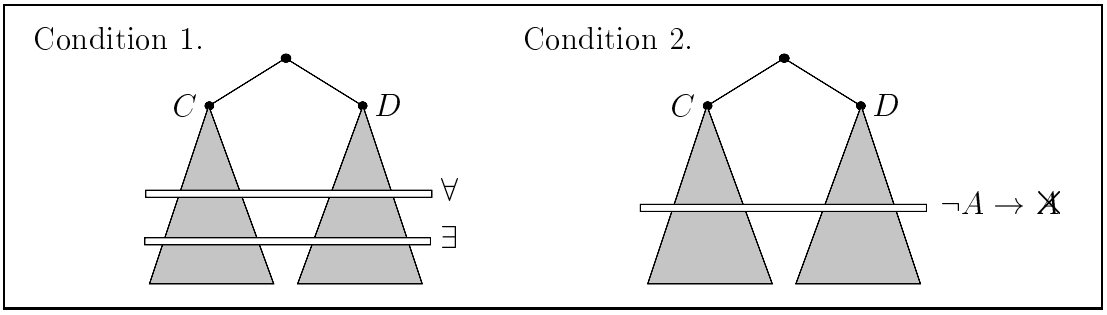


Figure 2: What nice concepts look like

Definition 5 *Let C be an $\mathcal{ALC}$-concept description in NNF. Then C is* nice *iff for every $d \in \mathbb{N}$ it holds that*

1. $|Q_r(C,d)| \leq 1$ *and*
2. $N_r(C,d)$ *does not contain a concept name and its negation.*

It remains to be shown that nice concepts as defined above in fact have the desired property. In preparation for this we firstly present a simple set-theoretic result which later on will allow us to reduce the number of existential restrictions computed in an approximation of certain nice concepts.

The distribution of a conjunction over a disjunction in the $\mathcal{ALC}$-normalization produces conjunctions of a very regular structure. As an example, consider the concept $E := (C_1 \sqcup C_2) \sqcap (D_1 \sqcup D_2)$ with $C_i := \exists r.C_i'$ and $D_j := \exists r.D_j'$. Assuming that all existential restrictions are $\mathcal{ALE}$-concepts, the normalization returns $\bigsqcup_{i,j}(C_i \sqcap D_j)$. The approximation algorithm then computes the lcs over every combination of existential restrictions from the four disjuncts. Nevertheless, every existential restriction in the result $approx_{\mathcal{ALE}}(E)$ either subsumes $\exists r.\mathsf{lcs}\{C_1', C_2'\}$ or $\exists r.\mathsf{lcs}\{D_1', D_2'\}$ because it corresponds to the lcs of a superset of one of the above sets. The following lemma shows that this subset-superset property can be generalized.

Lemma 6 *Let $m, n \in \mathbb{N}$. For every $i \in \{1,\ldots,m\}$ and $j \in \{1,\ldots,n\}$, let A_i and B_j be arbitrary finite sets, let $U_{ij} := A_i \cup B_j$, and let $u_{ij} \in U_{ij}$. Denote by U the set of all u_{ij}, i.e., $U := \{u_{ij} \mid 1 \leq i \leq m, 1 \leq j \leq n\}$. Then one of the following claims holds: either, for every i there exist elements $a_i \in A_i$ with $\{a_i \mid 1 \leq i \leq m\} \subseteq U$; or, for every j there exist $b_j \in B_j$ with $\{b_j \mid 1 \leq j \leq n\} \subseteq U$.*

For all $j \in \{1,\ldots,m\}$ and all $j \in \{1,\ldots,n\}$ consider arbitrary $u_{ij} \in U_{ij}$. Assume that the second claim for the sets $B_1,\ldots,B_n$ does not hold. Then there is one j' with $B_{j'} \cap U = \emptyset$, otherwise $b_{j'}$ could be chosen from this intersection to satisfy the claim. Since $u_{ij'} \in A_i \cup B_{j'}$ for all i it follows that $u_{ij'} \in A_i$ for all i, satisfying the first claim for $A_1,\ldots,A_m$. ∎

The choice of sets in the unions U_{ij} in the above lemma corresponds to tuples in the product $\{A_1, \ldots, A_m\} \times \{B_1, \ldots, B_n\}$. The claim can be generalized to n-ary products where every union corresponds to a tuple from $\{S_{11}, \ldots, S_{1k_1}\} \times \cdots \times \{S_{n1}, \ldots, S_{nk_n}\}$. A proof of this generalized version can be found in the technical report [6]. In the following lemma the above result is applied to the actual computation of the lcs.

Lemma 7 *For $1 \leq i \leq 2$, let C_i and D_i be $\mathcal{ALE}$-concept descriptions such that $C_1 \sqcap C_2 \sqcap D_1 \sqcap D_2$ is a nice concept. Then it holds that $\mathsf{lcs}\{C_i \sqcap D_j \mid i, j \in \{1, 2\}\} \equiv \mathsf{lcs}\{C_1, C_2\} \sqcap \mathsf{lcs}\{D_1, D_2\}$.*

The above claim can again be generalized to larger conjunctions. Let $1 \leq i \leq n$ and $1 \leq j \leq k_i$ and let C_{ij} be $\mathcal{ALE}$-concepts whose overall conjunction is nice. For every tuple $\bar{t} \in \{1, \ldots, k_1\} \times \cdots \times \{1, \ldots, k_n\} =: T$ denote by $C_{\bar{t}}$ the conjunction $\bigsqcap_{i=1}^{n} C_{i\bar{t}(i)}$. Then the least common subsumer $\mathsf{lcs}\{C_{\bar{t}} \mid \bar{t} \in T\}$ is equivalent to the conjunction $\bigsqcap_{i=1}^{n} \mathsf{lcs}\{C_{ij} \mid 1 \leq j \leq k_i\}$. The proof is analogous to the one shown above.

We are now ready to show that approximating nice concepts, as defined in Definition 5, can be simplified to a conjunction of approximations. For the sake of simplicity we restrict our attention to binary conjunctions. The proof for n-ary conjunctions is analogous.

Theorem 8 *Let $C \sqcap D$ be a nice $\mathcal{ALC}$-concept description. Then $approx_{\mathcal{ALE}}(C \sqcap D) \equiv approx_{\mathcal{ALE}}(C) \sqcap approx_{\mathcal{ALE}}(D)$.*

For the full proof refer to [6]. The claim is proved by induction over the sum of the nesting depths of $\sqcap$ and $\sqcup$ on every role-level in C and D. For the induction step a case distinction is made depending on whether C or D are conjunctions or disjunctions. If at least one concept description is a disjunction the approximation is defined as the lcs of all $\mathcal{ALC}$-normalized and approximated disjuncts (if one of the concepts is a conjunction, it firstly has to be distributed over the disjunction). The main idea then is to use Lemma 7 to transform single lcs calls of a certain form into a conjunction of lcs calls which eventually leads to the conjunction of the approximations of C and D.

Due to Theorem 8 it is now possible to split the computation of approximations into independent parts. Although this does of course not change the complexity class of the approximation algorithm it is still a significant benefit for practical applications. The improved approximation algorithm is displayed in Figure 3. The algorithm requires the unfolded input concept to be in NNF. In the first step the $\mathsf{c\text{-}approx}_{\mathcal{ALE}}$ function checks whether the approximation is trivial. If it is not the next step is to check whether the concept is nice. For nice concepts the $\mathsf{c\text{-}nice\text{-}approx}_{\mathcal{ALE}}$ function is invoked. For all other concepts the $\mathcal{ALC}$-normal form is computed lazily, i.e., the conjunctions are distributed over

Input: unfolded $\mathcal{ALC}$-concept description C already in NNF
Output: upper $\mathcal{ALE}$-approximation of C

c-approx$_{\mathcal{ALE}}$

1. If $C \equiv \bot$, then c-approx$_{\mathcal{ALE}}(C) := \bot$;
   if $C \equiv \top$, then c-approx$_{\mathcal{ALE}}(C) := \top$
2. If nice-concept-p(C) then return c-approx$_{\mathcal{ALE}}(C) :=$ c-nice-approx$_{\mathcal{ALE}}(C)$
3. Otherwise, transform the top-level of C into $\mathcal{ALC}$-normal form $C_1 \sqcup \cdots \sqcup C_n$ and return

   c-approx$_{\mathcal{ALE}}(C) :=$

$$\bigsqcap_{A \in \bigcap_{i=1}^{n} \mathsf{prim}(C_i)} A \quad \sqcap \quad \forall r.\mathsf{lcs}\{\mathsf{c\text{-}approx}_{\mathcal{ALE}}(\mathsf{val}_\mathsf{r}(C_i)) \mid 1 \le i \le n\} \ \sqcap$$
$$\bigsqcap_{(C_1',\ldots,C_n') \in \mathsf{ex}_\mathsf{r}(C_1) \times \cdots \times \mathsf{ex}_\mathsf{r}(C_n)} \exists r.\mathsf{lcs}\{\mathsf{c\text{-}approx}_{\mathcal{ALE}}(C_i' \sqcap \mathsf{val}_\mathsf{r}(C_i)) \mid 1 \le i \le n\}$$

c-nice-approx$_{\mathcal{ALE}}$

1. If $C \equiv \bot$, then c-nice-approx$_{\mathcal{ALE}}(C) := \bot$;
   if $C \equiv \top$, then c-nice-approx$_{\mathcal{ALE}}(C) := \top$
2. If $C = C_1 \sqcap \cdots \sqcap C_n$, then return
   c-nice-approx$_{\mathcal{ALE}}(C) := \bigsqcap_{i=1}^{n}$ c-nice-approx$_{\mathcal{ALE}}(C_i)$
3. Otherwise, return

   c-nice-approx$_{\mathcal{ALE}}(C) :=$

$$\bigsqcap_{A \in \bigcap_{i=1}^{n} \mathsf{prim}(C_i)} A \quad \sqcap \quad \forall r.\mathsf{lcs}\{\mathsf{c\text{-}nice\text{-}approx}_{\mathcal{ALE}}(\mathsf{val}_\mathsf{r}(C_i)) \mid 1 \le i \le n\} \ \sqcap$$
$$\bigsqcap_{(C_1',\ldots,C_n') \in \mathsf{ex}_\mathsf{r}(C_1) \times \cdots \times \mathsf{ex}_\mathsf{r}(C_n)} \exists r.\mathsf{lcs}\{\mathsf{c\text{-}nice\text{-}approx}_{\mathcal{ALE}}(C_i' \sqcap \mathsf{val}_\mathsf{r}(C_i)) \mid 1 \le i \le n\}$$

Figure 3: The improved algorithm c-approx$_{\mathcal{ALE}}$ and c-nice-approx$_{\mathcal{ALE}}$.

the disjunctions only for the current top-level. Then the c-approx$_{\mathcal{ALE}}$ algorithm proceeds as before. The c-nice-approx$_{\mathcal{ALE}}$ function for nice concepts works similar. Having treated the trivial cases, the second step is to test if the concept is a conjunction. In that case the approximation is obtained by splitting the concept conjunct-wise and making a recursive call for each conjunct. For all other nice concepts the approximation is computed as in c-approx$_{\mathcal{ALE}}$, besides the recursive calls refer to c-nice-approx$_{\mathcal{ALE}}$.

Observe that the test conditions for nice concepts can be checked in linear time once the concept description is unfolded and in NNF. Unfolding and trans-

forming the concept description into NNF always have to be performed to apply c-approx$_{\mathcal{ALE}}$, so that testing whether a concept is nice is hardly any extra effort when approximating a concept.

3.2 Approximating Nice Concepts in TBoxes

If an $\mathcal{ALC}$-TBox is to be translated into an $\mathcal{ALE}$-TBox, the concept description on the right-hand side of each concept definition has to be replaced by its approximation. For practical applications it is not feasible to perform such a translation in a naive way. The idea for optimizing this procedure is to re-use the approximation of a defined concept when approximating concept descriptions that in turn make use of this defined concept. More precisely, if we have already obtained the approximation of C and want to compute the approximation of, e.g., $(D \sqcap \exists r.C)$, we would like to be able to insert the concept description $\mathsf{approx}(C)$ directly into the right place in the concept description of $\mathsf{approx}(D \sqcap \exists r.C)$. Unfortunately, this approach does not work for arbitrary $\mathcal{ALC}$-concept descriptions due to possible interactions between different parts of the concept description. Nice concepts, however, are defined to rule out this kind of interaction. Hence, besides speeding-up the computation of a single approximation, the property of being a nice concept also is a prerequisite for caching and the re-use of already computed approximations. For example, if the defined concepts C_1, C_2, C_3 from the following TBox (with A, B and D as primitive concepts)

$$\begin{aligned} \mathcal{T} \quad = \{ \quad & C_1 = (\exists r.\neg A) \sqcup (\exists r.B), \\ & C_2 = \exists r.(\forall r.D \sqcup \neg E) \sqcap C_1 \sqcap \neg B, \\ & C_3 = \neg\,(\forall r.\exists r.(D \sqcap A) \sqcup \neg C_1 \sqcup \neg C_2) \quad \} \end{aligned}$$

are to be approximated and C_1 is approximated first, then this concept description can be re-used in subsequent approximations. If unfolded and transformed into NNF the concepts C_2 and C_3 are nice concepts. Hence, the approximation of C_2 is the conjunction of $\mathsf{approx}(\exists r.(\forall r.D \sqcup \neg E))$ and $\mathsf{approx}(C_1)$ and $\mathsf{approx}(B)$, where the already computed approximation of C_1 can be inserted directly. For C_3 we can re-use both approximations of C_1 and C_2 directly and only have to compute the approximation of $\exists r.\forall r.(\neg D \sqcup \neg A)$. Thus, the cost for approximating the entire TBox is reduced heavily.

4 Conclusion and Future Work

In this paper we have presented some first steps towards optimizing the computation of approximations. The main idea is to identify concepts that can be decomposed into parts which then can be approximated independently. These so-called nice concepts are structured in such a way that the top-level conjuncts

cannot interact with one another. Therefore, each conjunct can be approximated separately. Detecting nice concepts and approximating each of their conjuncts independently should be especially powerful in the context of translating entire $\mathcal{ALC}$-TBoxes into $\mathcal{ALE}$-TBoxes because it enables the direct re-use of already computed approximations and caching. Unfortunately, the conditions for nice concepts are very strict.

It is an open problem whether the rather strict conditions for nice concepts can be relaxed. To determine if independent approximation of nice concepts is a real benefit for practical applications, requires an implementation of modular approximation. Moreover, it is unknown if nice concepts occur frequently in application TBoxes.

Another open problem is whether the given conditions for nice concepts can be extended to the case where $\mathcal{ALCN}$-concept descriptions are approximated by $\mathcal{ALEN}$-concept descriptions.

References

[1] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

[2] F. Baader, R. Küsters, and R. Molitor. Computing least common subsumer in description logics with existential restrictions. In T. Dean, editor, *Proc. of IJCAI-99*, pages 96–101, Stockholm, Sweden, 1999. Morgan Kaufmann.

[3] F. Baader and A.-Y. Turhan. On the problem of computing small representations of least common subsumers. In *Proceedings of the German Conference on Artificial Intelligence, 25th German Conference on Artificial Intelligence (KI 2002)*, Lecture Notes in Artificial Intelligence. Springer–Verlag, 2002. To appear.

[4] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. In D. Fensel, D. McGuiness, and M.-A. Williams, editors, *Proc. of KR-02*, San Francisco, CA, 2002. Morgan Kaufmann Publishers.

[5] S. Brandt and A.-Y. Turhan. Using non-standard inferences in description logics — what does it buy me? In *Proceedings of the KI-2001 Workshop on Applications of Description Logics (KIDLWS'01)*, number 44 in CEUR-WS, Vienna, Austria, September 2001. RWTH Aachen. Proceedings online available from http://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-44/.

[6] S. Brandt and A.-Y. Turhan. An approach for optimizing $\mathcal{ALE}$-approximation of $\mathcal{ALC}$-concepts. LTCS-Report 02-03, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2002. See http://lat.inf.tu-dresden.de/research/reports.html.

[7] F. M. Donini, B. Hollunder, M. Lenzerini, A. M. Spaccamela, D. Nardi, and W. Nutt. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 2–3:309–327, 1992.

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

[9] R. Küsters. *Non-Standard Inferences in Description Logics*, volume 2100 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001. Ph.D. thesis.

[10] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

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

Corega Tabs: Incremental Semantic Composition

Kerstin Bücher, Günther Görz, and Bernd Ludwig
Computer Science Institute and FORWISS
FAU Erlangen, Germany
eMail: {buecher,goerz}@informatik.uni-erlangen.de,
bdludwig@immd5-wv.uni-erlangen.de

Abstract

In parsing natural language, incremental semantics composition is one of the most prominent issues. In the past, numerous approaches have been developed for assigning meaning to noun and verbal phrases and their complements and modifiers. Often, their inferential power is too low for practical applications or the expressiveness of the representation language leads to intractable inference procedures. As an answer to these problems, we discuss an approach that relies on Description Logics for handling this class of semantics construction. We show how a semantic knowledge base can be setup. We exploit the equivalence between Discourse Representation Structures limited to the expressiveness of ALC and ABoxes for validating DRS with respect to a given knowledge base.

1 Generic Dialogue Management in EMBASSI

The long-term goal of our research is to design and implement a generic dialogue system for rational (spoken) dialogues, which helps a user to achieve certain goals in terms of operations of a technical application system – e.g. an information system, a system for controlling devices, or any other kind of problem solving system. Among its design criteria are the ability to recognize users' intentions, to establish corresponding subgoals and control their processing. Furthermore, it shall enable mixed-initiative, flexible and cooperative conversations and provide a high level of robustness as well as scalability in the linguistic and application dimensions, which includes portability to new domains with as little effort as possible. It shall also be possible to integrate linguistic interaction with multimodal forms of input and output, as e.g. with graphical user interfaces and – by means of appropriate devices – the recognition of deictic actions.

To a large extent, our research and development work in the field of dialogue systems is done within the German joint project EMBASSI ("Elektronische Multimediale Bedien- und Service-Assistenz"), sponsored by the German Fed. Ministry of Research 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. Besides a chunk parser for spoken utterances, our contributions to this project are first, the dialogue manager, second, formal ontologies for several application domains and third, a language generation component to communicate system utterances to the user.

In this paper, we address in detail the issue of semantics construction during parsing natural language input[1]. As will be shown in section 3, the backbone of our incremental approach to composing semantic representations is λ-DRT [Fis96]: The parser builds Discourse Representation Structures (DRS)[KaR93] incrementally, and after each composition step, the satisfiability with respect to a given knowledge base is verified by an ABox consistency test. For this purpose, we exploit the fact that DRSs can be mapped onto ABoxes. In order to carry out all tasks necessary for semantics composition, we need a more general framework however, as there are several issues to consider which cannot be handled by using Description Logics [Don96] only.

2 Overview of the Levels of Utterance Analysis

Except for trivial cases, a direct mapping from a user utterance to a system command cannot be accomplished. In general, we have to take complex speech acts into account, where the interpretation of the utterance's propositional content is determined by its (local) linguistic-pragmatic context in the first place. This, in turn, is to a large extent influenced by (global) discourse-pragmatic features which provide constraints based on the dialogue history and the actual place of the utterance in the dialogue, as, e.g., being the expected answer to a question. Furthermore, the application provides further constraints by restricting the meaning of words and phrases to their particular use within a given thematic framework. Therefore, we have to distinguish several – interleaved – levels in the analysis of user utterances:

- Linguistic analysis on the utterance-local level, which in turn consists of several levels of syntactic and semantic construction (see section 3);
- Semantic evaluation, i.e. evaluation of semantic operators, reference resolution, and additional transformations of the logical form, augmented by specific computations;

[1] This includes some of the open questions mentioned in [Bue01].

- Application-domain specific specialization of the evaluated semantic representation (see[Lud02]);
- Discourse-pragmatic analysis - a proper function of the dialogue manager.

Syntactic Level. The syntactic level involves parsing a lattice of word hypotheses, using a two-step model for syntactic derivation. We describe syntactic analysis in section 3. In a first step, edges in the lattice are grouped in syntactic chunks [Abn91].[2] In a second step, these fragments are combined into bigger units obeying semantic and pragmatic constraints, the applicability of which is checked by valency and case frames of the lexical units.

Currently, we implement chunk grammars in a unification grammar formalism derived from PATR-II [Shi92] and augmented by DRS composition operators. In parsing with unification grammars, constraints are expressed as path equations. Instead of representing feature structures in a separate formalism, they could as well be expressed in DL. As a little experiment with CLASSIC showed, unification can then be achieved by means of the **same-as** construct for attributes, representing coreference. This construct is still a desideratum for grammar development with more powerful DL-systems.

Semantic Level. For semantic representation, we use the framework of Discourse Representation Theory (DRT)[3]. In particular, to provide a strictly compositional construction, our semantic representation formalism is λ-DRT, which combines the substitutional rigidity of λ-calculus with DRT [Kus96]. How the semantic construction is performed incrementally with the syntactic analysis is presented in section 3 Whereas the semantic representations of words are inserted during lexical scanning, their composition is performed by the execution of semantic operators which are attached to the rules of the chunk grammar.

Semantic evaluation. The DRS obtained by the semantic construction step has to be evaluated w.r.t. resolution of references, in particular of anaphors, and DRT-specific operators. For anaphor resolution, we developed a computational framework based on linguistic and pragmatic heuristics in [Fis96][4]. DRSs may contain logical operators, e.g. disjunction and conditional expressions and so called "duplex conditions" representing natural-language quantifiers. Evaluating such DRSs means to apply certain transformations to them. Disjunction will lead to two alternative DRSs. For quantifiers, the scope ambiguity problem can be resolved by applying the "Cooper storage" algorithm to several DRSs representing the different readings. The evaluation of particular natural-language quantifiers as "at-least n" will result in number restrictions. Furthermore, some

[2] This step is performed by a chart parser with a chunk grammar, working primarily with a head-driven bottom-up strategy.

[3] cf. [KaR93] for an introduction to Discourse Representation Theory

[4] For a general theoretical introduction with a similar computational solution, which covers also presupposition resolution, cf. vol. II of [Bla99]

generic computations like temporal and calendrical calculations to determine precise time and date specifications are performed in this step. Currently, all of these computations are implemented in a procedural way. With the availability of new tools such as TRIPLE [Sin02], we see an opportunity to specify these transformations in a uniform and more declarative way which fits very well with our underlying DL representation.

Application-domain specific specialization. In the next step, the evaluated semantic representation is transformed into a domain specific form where the general lexical concepts are replaced by domain concept structures according to the formal ontology of the application.

3 Incremental Semantic Composition

If we want human-computer-dialogues to be natural, we must allow humans to talk to the computer as they do to humans. Spontaneous speech often is incomplete or incorrect, full of interruptions and self-corrections, leading to an ungrammatical input to the parser. Additionally, given the error rates of speech recognizers, even with correct input the speech recognizer may produce an output which is not grammatical. Apart from this, parsing German input is difficult, since German is a language with fairly free word order, also allowing for discontinuous constituents. Therefore, the grammar cannot rely only on linear sequence as its main concept. We try to overcome these problems by designing a two-phase parsing process (as presented in [Bue02]). In this section we describe the two phases of parsing looking at the two levels of syntactic and semantic composition of words to chunks and, hopefully a proposition which - interpreted in its context - results in a system action.

3.1 Chunk Composition

The first phase works with a grammar that employs phrase structure rules to build small phrases, called chunks (similar to [Abn91]). A chunk consists of a head element C_h and not more than one other constituent C_f that is a possible filler of a free position in the head's (X-Bar-) structure.

$C \rightarrow C_1\ C_2$ where one (C_h; $h \in \{1, 2\}$) of the two categories is the head.

The filler usually is a complement (as is a noun phrase NP within a prepositional phrase PP) or a modifier (e.g. an adjective phrase $AdjP$ within an NP)[5]. A

[5]In generative grammar the term *c*omplement is used only for sisters of the lexical head. To avoid confusion we define a new term *c*omplifier that subsumes both complements and adjuncts.

chunk may also consist of only one constituent: $C \to C_1$. If C_1 is the head of the chunk and therefore a terminal lexical category, we get the semantics of C from the lexicon, where the semantic information is stored as a λ-DRS ([Kus96]) Δ. If C_1 is an expanded category[6] it contains the head of the chunk, and the semantics of C is inhibited from C_1. So, if there is only one symbol on the right side of the grammar rule, then the *extension* of the left side is determined as follows:

$$\text{ext}(C) := \begin{cases} \Delta & C_1 \text{ is the category and } \Delta \text{ is the DRS of the} \\ & \text{lexicon entry.} \\ \text{ext}(C_1) & \text{otherwise} \end{cases}$$

The *semantic head*[7] of the *chunk* is the one of its DRS:

$$\text{head}(C) := \text{head}(\text{ext}(C))$$

In case of a chunk consisting of a head C_h and another constituent C_f ($h \neq f \in \{1, 2\}$), C_f is related to the discourse referent d of C_h by a role R taken from the inventory of EuroWordNet (see [Lud02]). Syntactically, the combination of two categories to a chunk is determined by a grammar rule, which relates the two constituents via the role R. We then get the *extension* of the chunk by λ-composition of the DRSs of the two constituents (T is a DRS-variable):

$$\begin{aligned} \text{ext}(C) &:= (\lambda T.\Delta_k + T(d_2)) \left((\lambda T.\Delta_h + T(d_1)) \lambda x.y. \left[\frac{\emptyset}{R(x,y)} \right] \right) \\ &= \Delta_k + \Delta_h + \left[\frac{\emptyset}{R(d_1, d_2)} \right] \end{aligned}$$

So, when combining two elements, the parser checks the compatibility of the morphological features (e.g. agreement in case of the combination of a determiner with an NP) and merges their DRSs resulting in a DRS for the chunk that is consistent with the knowledge base (for details of the consistency check see 3.3). This way, each chunk gets an interpretation already at this early stage. If no further parsing is possible we thereby have means to interpret the whole utterance chunk by chunk.

For example, take the utterance "*Kommt Tatort im ZDF?*" from our Embassi application: To combine the preposition *im* and the NP-chunk *ZDF* which was build using the ($NP \to EN$)-rule we apply the following PP-rule[8]:

[6] An example would be a determiner phrase DP that is build from an NP that in turn is build from the lexical category N.

[7] Note that the syntactic head and the semantic head might not be the same; take the DP "*den Krimi*": the syntactic head of the DP is the determiner "*den*" but the semantic head is the noun "*Krimi*". Both heads of a phrase are defined in the prase structure rules.

[8] The fact that this utterance is a Yes/No-question is irrelevant to phase 1, but word order information (apart from intonation the only indicator of the type of speech act) is stored and made available when the pragmatics of the utterance is computed.

PP: P NP:
head = P:
role = has-value:
P morphfeat position = prepos,
P morphfeat kasrek = NP morphfeat case,
PP vpsynfeat clausetype = NP vpsynfeat clausetype,
PP = P:

$$\lambda P.NP.\left(\delta(P)\left(\delta(NP)\left(\lambda x.y.\left[\begin{array}{|l} \emptyset \\ \hline \texttt{has-value}(y,x) \end{array}\right]\right)\right)\right)$$

The PP-rule contains syntactic as well as semantic information about the chunk-combination. The DRS for the PP-chunk is achieved by λ-composition of the DRSs of *ZDF* and *im* taken from the lexicon related via the role `has-value`:

$$\left[\begin{array}{|l} i \\ \hline \texttt{im-SP}(i) \end{array}\right] + \left[\begin{array}{|l} l \\ \hline \texttt{TVStation1}(l) \\ \texttt{value}(l, ZDF) \\ \texttt{Name}(ZDF) \end{array}\right] + \left[\begin{array}{|l} \emptyset \\ \hline \texttt{has-value}(i,l) \end{array}\right] = \left[\begin{array}{|l} i\, l \\ \hline \texttt{TVStation1}(l) \\ \texttt{value}(l, ZDF) \\ \texttt{Name}(ZDF) \\ \texttt{im-SP}(i) \\ \texttt{has-value}(i,l) \end{array}\right]$$

After applying all phrase structure rules we get three chunks, i.e. the NP *Tatort*, the PP *im ZDF*, and the verb phrase VP *kommt*, that after this first phase have a semantic interpretation on their own. The interpretation of the whole utterance is derived by relating these chunks and their interpretation to each other. This is done in phase two.

3.2 Applying Case Frames to Chunks

The second phase is different from the first phase in that it relates chunks that do not need to be adjacent to each other, so the order of the constituents is not relevant but may be an indicator for preferred readings when disambiguation is called for. Phase 2 relies on a kind of dependency grammar that for each chunk of the first phase gives a list of possible syntactic functions the chunk may have:

$$C_1 \text{ has } C_2 \rightarrow \langle \texttt{synfunc} \rangle$$
$$\langle \text{constraint equation} \rangle$$

e.g.:

$$VP \text{ has } PP \rightarrow \texttt{adverbial}$$
$$NP \text{ has } PP \rightarrow \texttt{attribut}$$
$$VP \text{ has } NP \rightarrow \texttt{subject}$$
$$NP \text{ agr case} = \text{nom},$$
$$NP \text{ agr num} = VP \text{ agr num}.$$

The options are constrained by the morphological features of the chunk, e.g. an NP-chunk functions as subject only if it has nominative case.
For each chunk there is a case[9] frame for its semantic head that stores information about the valencies[10]. The valencies of each chunk are filled by combining it with other chunks, e.g. building a VP from a verb and an NP that functions as its direct object, or expanding a VP by an adverbial PP. The suitability of the combination of two chunks is determined by the semantic constraints of the application ontology. Take the case frame for *kommen*[11]:

infinitive: `kommen`

syntactic function	thematic role	lexical concept
`subject`	`involved-agent:`	`Program1`
`adverbial`	`involved-location:`	`TVStation1`

From the case frame we derive hypotheses about possible complifiers of a chunk using the syntactic functions. Whether a hypothesis is satisfiable is determined by the concepts of the chunks. If they fit (see 3.3), the DRS can be computed: For a semantic head C_h, its complifier C_k, and a theta role $R = \text{thema}(C_h, \texttt{synfunc})$ that C_k can fill, we get the extension of the modified chunk $\tilde{C_h}$ as follows:

$h := \text{head}(C_h)$, $k := \text{head}(C_k)$

$$\begin{aligned}\text{ext}(\tilde{C_h}) &= (\lambda T.\text{ext}(C_h) + T(\text{h}))(\lambda T.\text{ext}(C_k) + T(\text{k}))\left(\lambda x.y.\left[\frac{\emptyset}{R(x,y)}\right]\right)\\ &= \text{ext}(C_h) + \text{ext}(C_k) + \left[\frac{\text{h k}}{\text{thema}(C_1, \texttt{synfunc})(d_1, d_2)}\right]\end{aligned}$$

In our example, the VP *kommt* can be combined with the adverbial PP *im ZDF* since in the case frame of *kommen* there is a valency for an adverbial with the concept `location`. So we get

$$\left[\frac{i\,l\,k}{\begin{array}{l}\texttt{Run}(k)\ \texttt{TVStation1}(l)\ \texttt{involved-location}\\ \texttt{value}(l, ZDF)\ \texttt{Name}(ZDF)\ \texttt{im-SP}(i)\ \texttt{has-value}(i, l)\end{array}}\right]$$

After λ-composition of the DRS above with the DRS for *Tatort* we have a full DRS for our example utterance that is consistent with our knowledge base.

[9] The term *case* is used in the way of Filmore [Fil69] meaning thematic roles

[10] The term *valency* here is used in a broader sense: it includes not only obligatory elements needed to make a phrase syntactically complete; more than that, the case frames list all semantically and pragmatically suitable modifications and their syntactic representations, e.g. attributes for nouns or adverbials for verbs.

[11] The lexical concept is taken from EuroWordNet [Vos98]

3.3 Consistency Check

A DRS composed according to the algorithm outlined above has to be checked for consistency with respect to the given knowledge base. A DRS which passes the test, is called *admissible*. Given a DRS Δ_1 with discourse referent $d_1 = \text{head}(\Delta_1)$ and DRS Δ_2 with discourse referent $d_2 = \text{head}(\Delta_2)$ related via $R(d_1, d_2)$, we have to verify whether

$$\Delta = \Delta_1 + \Delta_2 + \left[\frac{\emptyset}{R(d_1, d_2)}\right]$$

is admissible. C_1 is the concept d_1 is an instance of, and analogously C_2 for d_2. Formally, Δ is *admissible* if and only if C_2 is a R-filler and C_1 is in the domain of R.

If we map Δ to an ABox A, A is inconsistent if Δ is not admissible. For a concept D with $\forall x : D(x) \leftrightarrow \neg C_2(x)$, we assume

$$C_1 \sqsubseteq \forall R.D,$$

If A was satisfiable, the following would hold:

$$\begin{aligned} d_1 \in C_1 \sqcap \forall R.C_2 \wedge d_1 \in C_1 \sqcap \forall R.D \quad \Leftrightarrow \\ d_1 \in \{x | C_1(x) \wedge \forall y(R(x,y) \rightarrow C_2(y) \wedge D(y))\} \end{aligned}$$

From $R(d_1, d_2)$ it follows that $C_2(d_2) \wedge D(d_2)$ holds in A – in contradiction to $C_2(d_2) \wedge \neg D(d_2)$. For R, two axioms are defined:

$$\begin{aligned} \exists R.\texttt{TOP} &\sqsubseteq D \\ \texttt{TOP} &\sqsubseteq \forall R.W \end{aligned}$$

If R was no restriction for C_1, C_1 is not in the domain of R, i.e. $C_1 \not\sqsubseteq W$. Assuming that Δ is still satisfiable, we get

$$d_1 \in \exists R.C_2 \Rightarrow d_1 \in \exists R.\texttt{TOP} \Rightarrow d_1 \in W$$

– a contradiction to $d_1 \in C_1 \not\sqsubseteq W$.

This DRS is domain independent in that it can be derived without context and without knowledge of the application. but therefore it also fails to connect the utterance to the application specific environment and discourse referents. How this connection is established is presented in [Lud02].

Acknowledgements

We are grateful to Yves Forkl, Martin Klarner and Peter Reiss for many discussions and valuable comments on earlier drafts of this paper.

References

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

[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.

[Bue01] K. Bücher, Y. Forkl, G. Görz, M. Klarner, and B. Ludwig, *Discourse and Application Modeling for Dialogue Systems.* Proc. KI-2001 Workshop on Applications of Description Logics, TU Wien, CEUR Proceedings, Vol. 44, 2001.

[Bue02] K. Bücher, M. Knorr, and B. Ludwig. *Anything to Clarify? Report Your Parsing Ambiguities!.* Proceedings of the 15th European Conference on Artificial Intelligence, July 22-26, 2002. Ed. Frank van Harmelen, Lyon, 2002, p. 465-469.

[Bla99] P. Blackburn and J. Bos, *Representation and Inference for Natural Language.* 2 vols., Saarbrücken, 1999. http://www.comsem.org

[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.

[Fil69] C. Fillmore. *Universals in Linguistic Theory*, chapter The Case for Case. Holt, Rinehart, and Winston, New York, 1969.

[Fis96] I. Fischer, B. Geistert, and G. Görz, *Incremental Semantics Construction and Anaphora Resolution Using Lambda-DRT.* Proceedings of DAARC-96 (Discourse Anaphora and Anaphor Resolution Colloquium, Ed. S. Botley and J. Glass), July 1996, Lancaster, p. 235-244.

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

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

[Lud02] B. Ludwig, K. Bücher, and G. Görz. *Corega Tabs: Mapping Semanatics onto Pragmatics*, this volume, 2002.

[Shi92] S. M. Shieber, *Constraint-Based Grammaar Formalisms. Parsing and Type Inference for Natural and Computer Languages.* A Bradford Book. Cambridge, Mass.: The MIT Press, 1992.

[Sin02] M. Sintek and S. Decker, *TRIPLE – A Query, Inference, and Transformation Language for the Semantic Web.* International Semantic Web Conference (ISWC), Sardinia, June 2002.

[Vos98] P. Vossen, editor. *EuroWordNet: A Multilingual Database with Lexical Semantic Networks.* Kluwer Academic Publishers, Dordrecht, 1998.

Model Construction for Configuration Design

Michael Eisfeld
Chair for Applied Computer Science in Civil Engineering,
Technical University Dresden, Germany
Email: michael.eisfeld@cib.bau.tu-dresden.de

Abstract

In this paper, we motivate the use of the expressive description logic $\mathcal{ALCQF}(\mathcal{D})$ with acyclic TBoxes for configuration design of structural systems. A solution to the design problem is automatically constructed from problem-specific constraints given in the ABox and the persistent conceptual description about the structure and its behaviour in the TBox. Furthermore, we argue that the configuration design problem, for which a solution is defined to be a canonical model of the constructed knowledge base, is decidable because the applied DL has the finite model property. We give an example of our approach by a simple routine design problem from practice.

1 Introduction

Conceptual design requires to construct configurations of structural elements, which channel the applied loads safely to the ground. Suitable alternatives have to be chosen according to the design constraints stated in the design brief. However, almost two-thirds of the solutions designed by engineers do not comply with the structural requirements and the given design constraints in practice. Therefore, structural engineers should be supported by knowledge-based problem solving methods. Their application reduces errors by exploiting persistent knowledge for similar design problem instances at the conceptual design stage. Conceptual design of simple structures shares many commonalities with configuration design because known parameterized elements have to be configured into a structural system with a valid topology. Consequently, problem solving methods for configuration design can be applied to conceptual design of such structural systems and thereby avoid the aforementioned deficiencies.

Configuration design is the task of searching for an assembly of predefined components as solution, which satisfies a set of requirements and obeys a set of constraints [18, 7, 8]. The problem of conceptually designing a structural frame from given design constraints by means of persistent structural knowledge about mereology and topology illustrates our chosen approach in the paper.

We focus on logic-based approaches because we have to employ implicitly represented knowledge. Different logic-based methods have been developed for configuration design and proposed for further research, see [17, 9, 4, 16]. Some of them have been even deployed in industrial settings [14]. In addition, hybrid approaches have been used, which apply different techniques for solving routine design tasks, in order to cover the complexity of given application domains [11]. However, they have a few drawbacks. Either the techniques are undecidable like those proposed in [10, 11], because they employ very expressive languages as first order predicate logic and a configuration language called BHIBS, or they have no declarative semantics for the used DL as in [14], which combines instance checking with procedural rules on different hierarchical levels. We propose a model construction approach for the chosen description logic $\mathcal{ALC}\mathcal{QF}(\mathcal{D})$, which possesses the finite model property and was first described in [1] and further developed in [9].

In our approach we represent conceptual knowledge about the structure and its requirements in the terminological component with acyclic TBoxes and knowledge about the behaviour and component attributes in distinct concrete domains, if they have different structural values. We use a arithmetical concrete domain for equilibrium statics and distinct concrete domains for numeric attribute values in order to represent properly the structural domain. We can thus describe that a component is in stable equilibrium because the sum of all terminal forces is zero. For example, the following concept describes that a local equilibrium of forces exists for a beam with an attribute of the interval $depth - range$ from the concrete domain for the dimension length:

$$Component \sqcap \exists beamDepth.depth{-}range \sqcap$$
$$\exists hasTerminal\ reaction1, hasTerminal\ reaction2,$$
$$hasTerminal\ action. + .$$

By means of the feature agreement we can define that a component and a system share a single terminal to their outer environment.

$$System \sqcap (InTerminal \downarrow Component\, InTerminal)$$

This is of particular interest for representing correctly the topology of a system. Problem-specific constraints and an actual configuration of the structure are represented in an ABox. The structural engineer starts with the specification

of the design constraints in the ABox. Afterwards the algorithm tries automatically to construct a canonical model by adding new assertions according to the defined knowledge base, in which a structural engineering expert afore has deschribed persistent conceptual knowledge about the structure's topology and mereology in the TBox. If a model can be constructed it is a solution to the configuration problem. The solution includes all necessary parts, completely instantiated components and their connected terminals, which secure valid load-paths through the structure. Different models can be built due to the specific constraints given in the ABox prior to the model construction process. We assume that the configuration problem is decidable if the inference problem of consistency testing of a TBox together with an ABox is decidable in our chosen DL. The description logic $\mathcal{ALC}\mathcal{QF}(\mathcal{D})$ is decidable because the fusion of the two decidable DLs $\mathcal{ALC}\mathcal{F}(\mathcal{D})$ [12] and $\mathcal{ALC}\mathcal{Q}$ [6] remains decidable according to [5].

Our paper is organized as follows. First, we introduce the employed description logic $\mathcal{ALC}\mathcal{QF}(\mathcal{D})$ with acyclic TBoxes, where we define sample concrete domains for behavioural constraints and concrete attributes. Second, we give a formal problem specification for model-based configuration design, where routine is defined by the availability of conceptual knowledge and behavioural constraints to the algorithm for the construction process at the outset. Afterwards, we give a short example from our structural engineering domain and describe briefly the reasoning service for constructing a model as solution on the given problem specification. Eventually, we conclude the paper.

2 The description logic $\mathcal{ALC}\mathcal{QF}(\mathcal{D})$

In this section we describe the decription logic $\mathcal{ALC}\mathcal{QF}(\mathcal{D})$. We give examples, why the increased expressivity of concrete domains and feature agreements is important for modelling a structural system and its mathematical equations from statics. We use the syntax and semantics given in [2].

Definition 1 (concrete domain) A *concrete domain* $\mathcal{D}$ is a pair $(\Delta_\mathcal{D}, \Phi_\mathcal{D})$, where $\Delta_\mathcal{D}$ is a set, the domain, and $\Phi_\mathcal{D}$ is a set of predicate names over $\Delta_\mathcal{D}$. Each predicate P from $\Phi_\mathcal{D}$ is associated with an arity n and a n-ary predicate $P^\mathcal{D} \subseteq \Delta_\mathcal{D}^n$. According to [3] a concrete domain $\mathcal{D}$ is called admissible iff:

- its set of predicate names is closed under negation and contains a name $\top_\mathcal{D}$ for $\Delta_\mathcal{D}$ and
- the satisfiability problem for finite conjunctions of predicates is decidable.

We restrict the use of concrete domains to admissible ones, for which we assume that $\bot_\mathcal{D}$ is the negation of the predicate $\top_\mathcal{D}$. Based on the former, we introduce

for the equilibrium equations a concrete domain $\Delta_R := \mathbb{Q}$ over the rational numbers, which was shown to be admissible in [13], and for each component attribute type a separate concrete domain Δ_{Ai} as proposed in [17]. The static equilibrium condition at terminals or for components can be specified on the basis of the following predicates:

- unary predicate $\top_R$ with $(\top_R)^R = \mathbb{Q}$ and a unary predicate $\bot_R$ with $(\bot_R)^R = \emptyset$,
- unary predicates $\leq, =$ and a binary predicate $=$ with the usual extension and
- a ternary predicate $+$ with $(+)^R = \{(q_1, q_2, q_3) \in \mathbb{Q}^3 \mid q_1 + q_2 = q_3\}$.

Additionally, we introduce the unary predicates *depth-range* and *force-range* for separate concrete domains of different structural values.

Definition 2 ($\mathcal{ALCQF}(\mathcal{D})$ syntax) Let C, R and F be disjoint sets of concepts, role and feature names and n a nonnegative integer. We call a composition $f_1 \cdots f_n$ of features a *feature chain*. If C and D are concepts of C, R is a role of R or F, $P \in \Phi_{\mathcal{D}}$ is a predicate of arity n and $u_1, \ldots, u_n$ are feature chains, then concepts can be formed according to the following rules:

$$\begin{aligned} C, D := \top \mid \bot \mid \neg C \mid \\ C \sqcap D \mid C \sqcup D \mid \\ \forall R.C \mid \exists R.C \mid \\ \geq n\, R.C \mid \ \leq n\, R.C \mid \\ \exists u_1, \ldots, u_n.P \mid u_1 \downarrow u_2 \mid u_1 \uparrow u_2, \end{aligned}$$

for which the feature chains u_i are abstract ones in the agreement and disagreement constructor and concrete feature chains for the existential predicate concept term, accordingly. We will use abbreviations for a feature chain $u = f_1 \cdots f_n$, $\exists u.C$ and $\forall u.C$ for $\exists f_1 \cdots \exists f_n.C$ and $\forall f_1 \cdots f_n.C$ respectively. We employ the usual set theoretic semantics for concepts.

Definition 3 ($\mathcal{ALCQF}(\mathcal{D})$ semantics) We use an *interpretation* $\mathcal{I} = (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$, which consists of a set $\Delta_{\mathcal{I}}$, namely the abstract domain, and an interpretation function $\cdot^{\mathcal{I}}$. $\Delta_{\mathcal{I}}$ is disjoint from $\Delta_{\mathcal{D}}$. The interpretation function maps each concept name C to a subset $C^{\mathcal{I}}$ of the abstract domain, each role name R to a subset of $R^{\mathcal{I}}$ of $\Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}}$, and each feature name f to a partial function $f^{\mathcal{I}} \colon \Delta_{\mathcal{I}} \to \Delta_{\mathcal{D}} \cup \Delta_{\mathcal{I}}$, which we write in the extensional form $(a^{\mathcal{I}}, x^{\mathcal{I}}) \in f^{\mathcal{I}}$. For a feature chain u, the semantics is given by the composition of the partial functions interpreting its components, i.e. $u_1^{\mathcal{I}} = f_n^{\mathcal{I}}(\cdots f_1^{\mathcal{I}}(a^{\mathcal{I}}) \cdots)$. If the symbols

are defined as in Definition 2, we extend the interpretation function to complex concept terms as follows:

$$
\begin{aligned}
(C \sqcap D)^{\mathcal{I}} &:= C^{\mathcal{I}} \cap D^{\mathcal{I}}, (C \sqcup D)^{\mathcal{I}} := C^{\mathcal{I}} \cup D^{\mathcal{I}}, (\neg C)^{\mathcal{I}} := \Delta_{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid \exists b \in \Delta_{\mathcal{I}} : (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid \forall\, b : (a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
(\geq n\, R.C)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid\mid \{b \in \Delta_{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C)^{\mathcal{I}}\}| \geq n\} \\
(\leq n\, R.C)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid\mid \{b \in \Delta_{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C)^{\mathcal{I}}\}| \leq n\} \\
(\exists u_1, \ldots, u_n.P)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid \exists x_1, \ldots, x_n \in \Delta_{\mathcal{D}} : \\
&\qquad (a, x_1) \in u_1^{\mathcal{I}} \wedge \cdots \wedge (a, x_n) \in u_n^{\mathcal{I}} \wedge (x_1, \ldots x_n) \in P^{\mathcal{D}}\} \\
(u_1 \downarrow u_2)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid \exists b \in \Delta_{\mathcal{I}} \colon b = u_1^{\mathcal{I}} = u_2^{\mathcal{I}}\} \\
(u_1 \uparrow u_2)^{\mathcal{I}} &:= \{a \in \Delta_{\mathcal{I}} \mid \exists b_1, b_2 \in \Delta_{\mathcal{I}} \colon u_1^{\mathcal{I}} = b_1, u_2^{\mathcal{I}} = b_2 \wedge b_1 \neq b_2\}
\end{aligned}
$$

We use the standard notion of a *model* for a concept C and apply the standard reductions between the inference problems as can be found in [2].

We employ a TBox for specifing the conceptual model of the structure, the persistent requirements from statics and the loads on the structure. This also includes necessary parts and terminal connections.

Definition 4 (TBox) An *terminological axiom* φ is a concept definition $C \doteq D$ or a concept specialization $C \sqsubseteq D$, where C is a concept name and D is a complex concept description. A finite set of such axioms is called a *TBox* $\mathcal{T}$. We use the standard semantics for TBoxes. An interpretation $\mathcal{I}$ satisfies an axiom of the form $C \doteq D$ and $C \sqsubseteq D$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$ and $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, respectively. Eventually, we say that an interpretation $\mathcal{I}$ is a *model* for the the TBox $\mathcal{T}$ iff it satisfies all φ in $\mathcal{T}$. We restrict $\mathcal{T}$ to include only acyclic axioms. Thus, we can expand the given TBox before starting the reasoning process.

We use an ABox for the task-specific constraints and the solution structure, which is built up by the model construction process.

Definition 5 (ABox) Let O_D and O_A be disjoint sets of so-called *concrete* and *abstract objects*. If C, R, f and P are defined as in Definiton 2, a and b are elements of O_A and $x, x_1, \ldots, x_n$ are elements of O_D, then the following expressions are an *assertional axiom* ϕ:

$$a : C,\ (a : b) : R,\ (a, x) : f,\ a \neq b,\ (x_1, \ldots, x_n) : P.$$

We call a finite set of such axioms an *ABox* $\mathcal{A}$ and extend the notion of an interpretation to the assertional component by mapping every object name from

O_A to an element of $\Delta_\mathcal{I}$ and every object name from O_D to an element of $\Delta_\mathcal{D}$, for which the unique name assumption is not imposed. An interpretation $\mathcal{I}$ satisfies an assertion

$$\begin{aligned} a : C \text{ iff } a^\mathcal{I} \in C^\mathcal{I}, \\ (a, b) : R \text{ iff } (a^\mathcal{I}, b^\mathcal{I}) \in R^\mathcal{I}, \\ (a, x) : f \text{ iff } (a^\mathcal{I}, x^\mathcal{I}) \in f^\mathcal{I}, \\ a \neq b) \text{ iff } a^\mathcal{I} \neq b^\mathcal{I}, \\ (x_1, \ldots, x_n) : P \text{ iff } (x_1^\mathcal{I}, \ldots, x_n^\mathcal{I}) \in P^\mathcal{D}. \end{aligned}$$

An interpretation $\mathcal{I} \models \mathcal{A}$, for which we call $\mathcal{I}$ a *model* of the ABox, iff $\mathcal{I} \models \phi$ for all $\phi \in \mathcal{A}$. $\mathcal{A}$ is *consistent* iff it has a model. The canonical model is built up from $\mathcal{A}$ by determining a logical structure from all assertions of the final ABox, which were added by rules application during the construction process. Note that we can construct different models, when we specify different task-dependent constraints in $\mathcal{A}$ at the beginning of the model finding process.

3 Configuration design problem

In this section we start with defining the configuration design problem before we give an example from our structural engineering domain. We use the notion of constructing an admissible configuration in the ABox from a conceptual description in the TBox as first defined in [1].

Definition 6 (configuration design problem) A *configuration design problem* $P_C = (\mathcal{T}, \mathcal{A}_0)$ consists of a TBox $\mathcal{T}$ and an initial ABox $\mathcal{A}_0$. In the TBox a conceptual description of possible configurations is decribed. The initial ABox specifies the design constraints, the goal object and other necessary information for the model construction process. A solution can be constructed if a *finite model* for P_C exists. This model represents an textitadmissible configuration.

Depending on the stated design constraints in the initial ABox, the algorithm computes different models for a fixed TBox by introducing new assertions into the ABox. The algorithm makes thereby all implicit knowledge from $\mathcal{T}$ explicit. It computes first a complete ABox $\mathcal{A}$. Afterwards the canonical model can be built on the domain. The task-dependent requirements stated in $\mathcal{A}_0$ reduce thereby the possible solutions, which can be generated on $\mathcal{T}$. For computing a canonical model as solution to our configuration problem P_C, we use an algorithm as described in [13], which uses concept satisfiability for testing consistency of $\mathcal{T}$ and $\mathcal{A}$.

Example Our configuration problem is a simple frame design. In $\mathcal{T}$ all requirements regarding necessary parts, the layout of the components and the allowable concrete domains, which limit structural values to certain intervals or single concrete objects, are defined. Some problem instance specific constraints on the solution together with the frame object, which has to be configured, are stated in $\mathcal{A}_0$ before the construction process starts.

We begin with the definition of the system and descend the hierarchy top-down from the frame to the load definition. All frames have some common components. Two overall frame solutions are possible depending on the constraint in $\mathcal{A}_0$. The structural engineer is mainly interested in a valid topology. Numeric values are usually not considered at the conceptual design stage, unless they violate stated geometrical or structural constraints. If this is the case, the conceptual description of single component parameters or terminals has to be rewritten in $\mathcal{T}$. We do not take this case into account because we assume a fixed conceptual description during the solution process.

The following concept definitions draw also on the aforehand defined concrete domains from definition 1. The equilibrium of the components' terminals is specified by the predicates $+, =$ and $\leq$ over the concrete domain for the dimension force, while the predicate $\leq$ is defined over a separate concrete domain for the dimension length. The predicate *depth-range* and *force-range* denote unary predicates that restricts the numeric value for the dimension length and force to be in the interval of $I_1 = [0.4, 0.7]$ and $I_2 = [5, 20]$. The *force-range* predicate restricts the applicability of the solution method to reasonable loading levels. The lower bound of the *depth-range* constraint is stipulated in the code and the upper bounded by economical considerations.

$$
\begin{aligned}
StructuralSystem \doteq\ & Frame \sqcap \\
& (InTerminal \downarrow \\
& Beam1\ InTerminal) \sqcap \\
& (OutTerminal \downarrow \\
& Column1\ OutTerminal) \sqcap \\
& (OutTerminal \downarrow \\
& Column2\ OutTerminal) \sqcap \\
& (= 1\ loadedBy.Load) \\
Frame \doteq\ & ((\neg Slender \sqcup RigidFrame) \sqcup \\
& (\neg Cheap \sqcup SimpleFrame)) \sqcap \\
& (= 1\ hasPart.Beam) \sqcap \\
& (= 2\ hasPart.Column) \sqcap \\
& (= 2\ hasPart.Support) \sqcap \\
& (= 4\ hasPart.Connection)
\end{aligned}
$$

$$
\begin{aligned}
RigidFrame &\doteq (= 4\ hasPart.Rigid) \sqcap \\
&\quad (= 2\ hasPart.Fixed) \\
SimpleFrame &\doteq (= 2\ hasPart.Rigid) \sqcap \\
&\quad (= 2\ hasPart.Hinged) \sqcap \\
&\quad (= 2\ hasPart.Simple) \\
Component &\sqsubseteq VerticalMember \sqcup Beam \\
VerticalMember &\doteq (Column \sqcup Connection \sqcup Support) \sqcap \\
&\quad \exists hasTerminal\ action, \\
&\quad hasTerminal\ reaction\ . = \\
Beam &\doteq Bending \sqcap \\
&\quad \exists beamDepth.depth{-}range \sqcap \\
&\quad \exists beamLength.\ \leq_{10} \sqcap \\
&\quad \exists hasTerminal\ reaction1, \\
&\quad hasTerminal\ reaction2, \\
&\quad hasTerminal\ action.+ \\
Column &\doteq Compression \sqcap \\
&\quad \exists columnDepth.depth{-}range \sqcap \\
&\quad \exists columnLength.\ \leq_4 \sqcap \\
Support &\doteq Simple \sqcup Fixed \\
Connection &\doteq Hinged \sqcup Rigid \\
Terminal &\sqsubseteq \neg Component \sqcap \neg StructuralSystem \sqcap \neg Load \sqcap \\
&\quad (InputTerminal \sqcup OutputTerminal) \\
InputTerminal &\doteq \exists action.\ \geq_0 \\
OutputTerminal &\doteq \exists reaction.\ \geq_0 \sqcap \\
&\quad (= 1\, connected.InTerminal) \\
Load &\doteq \exists concentratedForce.force{-}range \sqcap \\
&\quad \exists hasTerminal\ action.\ \geq_0
\end{aligned}
$$

The task specific assertions are given in the initial ABox $\mathcal{A}_0$.

$$
\begin{aligned}
&\{FRAMEI\colon Frame, FRAMEI\colon Slender, \\
&VLOAD\colon Load, \\
&(FRAMEI, VLOAD)\colon loadedBy, \\
&(BEAMI, x_1)\colon beamLength, (x_1)\colon =_8, \\
&(BEAMI, x_2)\colon beamDepth, (x_2)\colon =_{0.5}, \\
&(COLUMN1, x_3)\colon columnLength, (x_3)\colon =_3,
\end{aligned}
$$

$(FRAMEI, TERM1)$: $InTerminal$,
$(VLOAD, TERM2)$: $hasTerminal$,
$(TERM2, TERM1)$: $connected$,
$(VLOAD, x_4)$: $concentratedForce$, (x_4): $=_{10}\}$

We introduce $x_1, \ldots, x_3$ as concrete objects and x_4 as a concrete object over another concrete domain, respectively. The equality predicates restrict structural values to single objects from the concrete domains.

An algorithm for testing consistency of the configuration problem P_C can compute a complete ABox $\mathcal{A}$ [15]. If the configuration problem is inconsistent, namely not clash free, there might be two distinct cases. First, constraints were stated in $\mathcal{A}_0$, which are inconsistent on $\mathcal{T}$ without introducing new objects during the construction process at all, being the case for contradictory task-specific constraints in $\mathcal{A}_0$. Second, no solution to P_C can be found on the completed ABox $\mathcal{A}$. If the ABox is clash free, a canonical model can be constructed from the generated object names in $\mathcal{A}$. We give only some typical excerpts from the whole model due to space restriction and restrict us to one half of the frame because of symmetry. We do not specify values for the concrete objects from the domain Δ_R because the actual force distribution is not taken into account by practitioners in the conceptual design process. The real force distribution is usually afterwards calculated by structural analysis tools.

$\Delta_{\mathcal{I}} = \{FRAMEI, BEAMI, COLUMN1, FIXED1, RIGID_i, TERM_j, \ldots\}$,
$\Delta_R = \{force1, \ldots, force20\}, \Delta_{A1} = \{10\}, \Delta_{A2} = \{0.5, 3, 8\}$,
$StructuralSystem^{\mathcal{I}}, RigidFrame^{\mathcal{I}} = \{FRAMEI\}$,
$Beam^{\mathcal{I}} = \{BEAM1\}, Column^{\mathcal{I}} = \{COLUMN1\}$, etc.,
$InputTerminal^{\mathcal{I}} = \{TERM1, \ldots, TERM9\}$,
$OutputTerminal^{\mathcal{I}} = \{TERM2, \ldots, TERM11\}$,
$InTerminal^{\mathcal{I}} = \{(FRAMEI, TERMI), (BEAMI, TERMI), \ldots\}$, etc.,
$hasPart^{\mathcal{I}} = \{(FRAMEI, BEAMI), (FRAMEI, COLUMN1), \ldots\}$,
$hasTerminal^{\mathcal{I}} = \{(BEAMI, TERM1), (COLUMN, TERM3), \ldots\}$,
$action^{\mathcal{I}} = \{(TERM1, force1), (TERM3, force3), \ldots\}$,
$reaction^{\mathcal{I}} = \{(TERM2, force2), (TERM4, force4), \ldots\}$,
$connected^{\mathcal{I}} = \{(TERM3, TERM4), (TERM2, TERM1), \ldots\}$.

The constructed model $\mathcal{I}$ from the complete and clash free ABox is a solution, which describes the actual assembly in terms of the required parts being the specific components and terminals, the components attributes as well as the resulting topology.

4 Conclusion

We have shown that configuration design problems from the domain of structural engineering can be solved by model construction in a description logic like $\mathcal{ALCQF}(\mathcal{D})$, which possesses the finite model property. We focused on the computation of parts, namely components and terminals of the structural system on different hierachical levels, and of a layout that ensures channeling the loads within the structure safely to the ground. Especially, the topological requirements on the structure required feature aggreement in the DL. As far as we know, regular configuration languages cannot represent topological requirements yet, which are important if the structure of the configuration cannot be considered fixed prior the model construction approach.

In addition, we found that only a subset of possible concept expressions,which are defined by the syntax of $\mathcal{ALCQF}(\mathcal{D})$, was used for the definition of the structural description in the TBox. It might be possible to take such restriction into account for increasing the expressiveness of the description logic but still obtaining decidable inference problems. Hence, the model construction approach extends to subsets of even more expressive description logics with infinite models, as long as the employed subset used for the formulation of the configuration problem has the finite model property.

Because intuitively many finite solutions exist for usual configuration design problems from engineering domains, it might be a further research direction to identify more clearly a decidable subset of very expressive decription logics, which cover the representation requirements from engineering configuration domains.

References

[1] F. Baader, M. Buchheit, and B. Hollunder. Cardinality restrictions on concepts. In *Proceedings of the German AI Conference, KI'94*, volume 861 of *Lecture Notes in Computer Science*, pages 51–62, Saarbrücken (Germany), 1994. Springer–Verlag.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P Patel-Schneider. *Description Logic Handbook*. Cambridge University Press, 2002.

[3] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. DFKI Research Report RR-91-10, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, 1991.

[4] F. Baader and P. Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proceedings of the 16th German AI-*

Conference, GWAI-92, volume 671 of *Lecture Notes in Computer Science*, pages 132–143, Bonn (Germany), 1993. Springer–Verlag.

[5] F. Baader, C. Lutz, H. Sturm, and F. Wolter. Fusions of description logics and abstract description systems. *Journal of Artificial Intelligence Research (JAIR)*, 16:1–58, 2002.

[6] F. Baader and U. Sattler. Expressive number restrictions in description logics. *Journal of Logic and Computation*, 9(3):319–350, 1999.

[7] D. C. Brown. Defining configuring. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 12(4):301–305, March 1997.

[8] D. C. Brown and B. Chandrasekaran. *Design Problem Solving, Knowledge Structures and Control Strategies*. Morgan Kaufmann Publishers, 1989.

[9] M. Buchheit, R. Klein, and W. Nutt. Constructive problem solving: A model construction approach towards configuration. DFKI technical memo TM-95-01, Deutsches Forschungszentrum für Künstliche Intelligenz Saarbrücken, January 1995.

[10] G. Friedrich and M. Stumptner. Consistency-based configuration. In *Proceedings of the AAAI '99 Workshop on Configuration*, 1999.

[11] O. Hollmann, Th. Wagner, and A. Günter. Engcon: A flexible domain-independent configuration engine. In *Proceedings of the ECAI-Workshop Configuration, ECAI-2000*, 2000.

[12] C. Lutz. Reasoning with concrete domains. In Thomas Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence IJCAI-99*, pages 90–95, Stockholm, Sweden, July 31 – August 6, 1999. Morgan-Kaufmann Publishers.

[13] C. Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen, 2001.

[14] D. McGuinness and J. Wright. An industrial strength description logic-based configurator platform. *IEEE Intelligent Systems*, 13(4):69–77, July 1998.

[15] R. Möller. Expressive Description Logics: Foundations for Practical Applications. Faculty of Computer Science, University of Hamburg, 2000. Habilitation.

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

[17] C. Schröder, R. Möller, and C. Lutz. A partial logical reconstruction of PLAKON/KONWERK. In *Proceedings of the Workshop on Knowledge Representation and Configuration WRKP-96*, pages 55–64, 1996.

[18] B. Wielenga and G. Schreiber. Configuration-design problem solving. *IEEE Intelligent Systems*, 12(2):49–56, March 1997.

Design and Usage of an Ontology for Supply Chain Monitoring

A. Huber, G. Görz,
R. Zimmermann, S. Käs, R. Butscher, F. Bodendorf
University of Erlangen-Nuremburg, Germany
Email: {arhuber, goerz}@informatik.uni-erlangen.de,
roland.zimmermann@wiso.uni-erlangen.de

Abstract

We present an agent-based supply chain monitoring system for tracking orders and their related suborders. We introduce an ontology to enable the necessary communication between the agents. Its design and implementation are discussed in detail.

1 Introduction

Fulfillment processes in a supply chain may show different kinds of irregularities and disruptions. Irregularities like delays in a production process are the result of variances in processing times and processing quality. The orders issued by different supply chain partners are often linked, e.g. an order for a car induces suborders for parts from suppliers (tires, seats, etc.). The tighter integrated the partner's supply chain processes are the greater will be the impact of disruptions on the fulfillment processes. Existing systems for order tracking typically generate standard messages for each order at certain intervals or milestones. As a result, large databases with information on orders are filled with data that are in most cases not related to the serious problems mentioned above. Only a small percentage of the orders encounters problems during fulfillment. Although information might be available, it is generally communicated too late, the content will often not match the needs of a decision maker to react to the problem and is lost in the large set of data.

Supply chain monitoring solutions for tracking orders need to be analyzed in the context of a domain. A typical logistics scenario consists of a variety of manufacturers, suppliers, and logistics service providers. Figure 1 illustrates

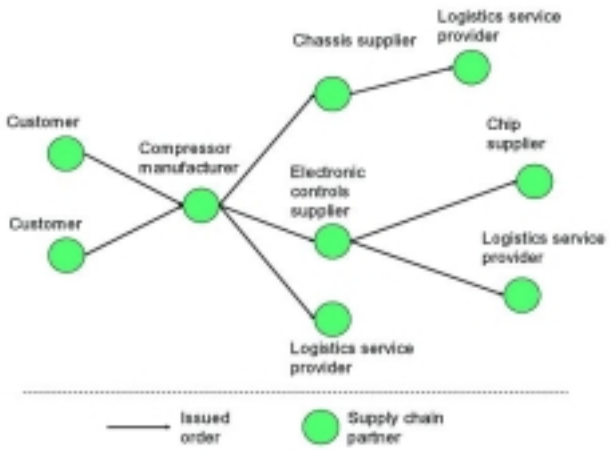


Figure 1: Supply Chain scenario.

a scenario of a compressor supply chain. The links between the organizations represent the issued orders and suborders resulting from customer orders.

We developed a decentralized agent-based concept for supply chain monitoring that differs from traditional isolated and centralized tracking systems that are widely implemented within logistics service provider networks [6] and similar approaches of production control [7]. The solution focuses on critical orders, allows for real-time tracking of orders across the supply-chain, and offers the opportunity to react in a timely manner to unforeseen events during order fulfillment. Our supply chain monitoring approach focuses on tracking of individual orders already issued. The aim is to gather information on orders and their related suborders with the help of software agents.

Two layers characterize such an agent architecture. The first layer is represented by a discourse agent which is concerned with communicating the tracking information across the supply chain by interacting with the different agent platforms of the supply chain participants. The second one, the application layer, has to gather and aggregate the tracking information from which actions can be derived. A detailed description of the functionality and interaction of these agents is presented in [10]. To enable the communication of information between the agents an ontology for supply chain monitoring is designed.

2 Ontology-Based Agent Communication

Agent-to-agent communication is the key to realize the potential of the agent paradigm and is a requirement for cooperation. Agents use an Agent Communication Language (ACL) to communicate information and knowledge about the domain. Systems which communicate and work together must share an ontology

which can be implicit or explicit. Implicit ontologies are typically represented only by procedures whereas explicit ones are (ideally) given as declarative representation in a well-defined knowledge representation language. It introduces the concepts and relations needed to exchange messages about application relevant information. Once an ontology has been incorporated into the communication, it can be used to construct the message content, one of the key points for communication. The message content can have different granularity with respect to the ontologies level of abstraction. In communication with an agent, another agent can ask about both the general knowledge and the knowledge of a specific topic.

On the one hand, the ontology determines what is the possible content of a message and on the other hand is used to analyse the messages that should lead to appropriate tasks or actions. For this purpose a reasoning module is needed. The agent must verify the content with regard to misconceptions. The reasoning module is used to check if all concepts and relations exist in the ontology and if they are used properly. If the message contains a question about some individuals the agent has to verify their existence. In both cases, if an error is detected, the resulting answer would be negative and should be explained further in order to be cooperative. In this response the agent must revise the misconception. If the content is correct, the agent interprets the message and tries to find an answer. An answer can be a response containing the requested information or some course of actions to fulfil a postulated goal.

Different ontology description languages exist to formalize ontologies. The supply chain tracking ontology is modeled using the Ontology Inference Layer (OIL) [4]. OIL unifies three important aspects provided by different communities: it inherits the formal semantics and reasoning support from Description Logics, incorporates the essential modeling primitives of Frame-based systems, and uses existing Web Standards by providing XML and RDF based syntax [1]. The reasoning is done with Racer [2] a tableau based reasoning module.

3 The Supply Chain Monitoring Ontology

3.1 Methodological Approach

To design an ontology, it is important to define the functionality of the ontology and to characterize the users of the ontology [8]. The ontology represents an integrated supply chain model spanning all processes, organizational units, and objects involved in the scenario. The supply chain monitoring ontology defines all concepts necessary for the supply chain environment as well as basic concepts of tracking data. It is not the aim of this ontology to provide a formal representation of all the aspects of supply chain management. Its purpose is to model the characteristic features of tracking orders in a supply-chain. Therefore the

modelled concepts are restricted to this domain. Based on this understanding, a rough first draw of the ontology can be derived that is iteratively refined step by step. There are several possibilities to derive this initial design. Holsapple and Joshi [3] identify five basic approaches to ontological design: inspiration, induction, deduction, synthesis, and collaboration. Combinations of these approaches are possible. For the design of the supply chain monitoring ontology a combination of the inductive and the synthetic approach is used. The inductive approach is characterized by observing, examining, and analysing (a) specific case(s) in the domain of interest and applying this specific case(s) to other cases in the same domain [3]. The intitial ontology is based on a scenario derived from data of a business partner. This basic scenario consisted of one customer, one vendor and one logistics service provider. In a top-down process the most general concepts of this specific scenario were defined and subsequently refined. In a second step, the ontology was enlarged to incorporate scenarios including several manufacturers, suppliers and logistics service providers. However, not all concepts in the ontology are new. Concepts from existing ontologies are adopted and synthesized into the supply chain monitoring ontology (synthetic approach). To model this ontology the Enterprise Ontology [9] is partially reused. It specifies a wide variety of concepts from the domain of enterprises. As the supply chain monitoring ontology belongs to the same domain, the Enterprise Ontology already provides some important general concepts that are also necessary for the supply chain scenario. The main benefit of using the Enterprise Ontology is to allow different multi-agent systems that may be concerned with varying aspects of supply chain management (e.g. procurement planning vs. order tracking) to communicate on a generic level as long as they commit themselves to the same top-level-ontology (e.g. the Enterprise Ontology). However, besides refinements of existing concepts, some important high-level concepts for the supply chain domain are missing and have been added, e.g. the concept of an order.

3.2 Concepts

The concepts of the supply chain monitoring ontology described are needed to reflect the scenario of a supply chain and to express the tracking data in a formalized mode. To represent the scenario the fulfilment process can be described as the interaction between three main concepts: *Actors*, *Activities/Processes* and *Orders*. Linked to the concept of an *Order* are the different tracking data types. An *Actor* issues an *Order* which is then received by another *Actor* in the supply chain. More *Actors* can be involved in the fulfilment process if further *Orders* have to be issued to be able to fulfil the first *Order*. This is the case if a customer orders a product from a manufacturer who then needs to order components from his suppliers for the assembly of this product. When issuing the *Order*, a sequence of *Activities/Processes* is triggered which has to be performed

to fulfil the *Order*. These *Activities* are carried out by the respective *Actors*. As the fulfilment process is dynamic the concept of *Time* is relevant in the context of supply chain monitoring. The time axis is necessary for tracking the status of the fulfilment process as the *Activities* are carried out over a period of time.

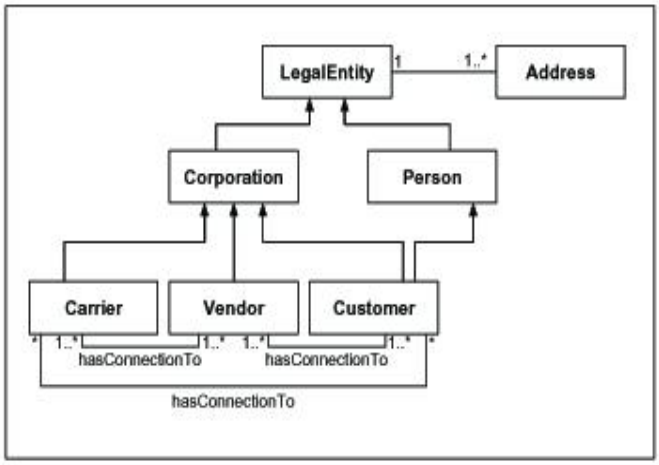


Figure 2: Legal Entity.

The central concept concerning *Actors* is *LegalEntity* (see fig. 2). A *LegalEntity* is recognized as having rights and responsibilities in the world by large and by legal jurisdiction in particular [9]. It includes *Person* and *Corporation*. *Vendor* and *Carrier* are both *Corporations*. The *Customer* can either be a *Person* or a *Corporation*.

The central concept *Activity* (see fig. 3) represents the generalization of all actions that need to be executed within the fulfillment process. On the one hand *Activity* is needed to describe a specific scenario while on the other hand the fulfillment of an *Activity* can indicate the achievement of a *Milestone* and is therefore also important for representing tracking data.

For the supply chain monitoring ontology eleven specific activities can be identified: *OrderReceipt*, *ConfirmationOfOrder*, *ProductionOrderOpening*, *ProductionPlanning*, *Manufacturing*, *QualityAssurance*, *Picking*, *Packaging*, *OutgoingGoods*, *Handling*, *Delivery*. The execution of an *Activity* leads to a specific *Effect*, which consists of a *StateOfAffairs* that must hold true at a point of time (which is specified by the *EffectWhenHold* concept). An *Activity* has a begin and an end date. The state of the *Activity* at a point of time is described by the *ActivityState*. The fulfillment process is a sequence of *Activities* that can usually only be performed one after another. Therefore most *Activities* have as a *Precondition* the execution of another *Activity*. An *Activity* is performed by an *Actor*.

The concept of an *Order* is another basic concept to model a supply chain

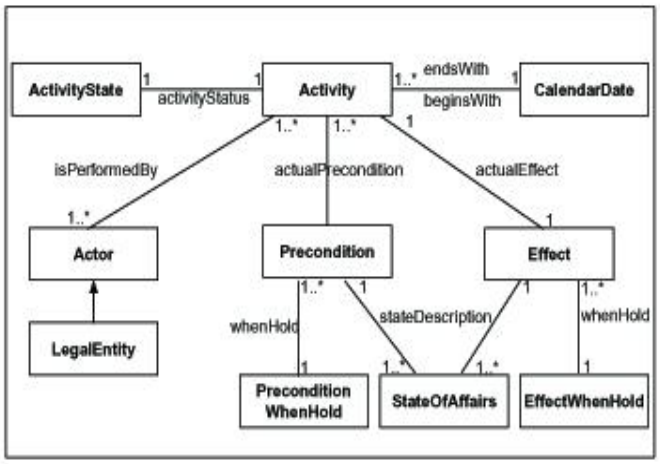


Figure 3: Activity concept.

scenario (see fig. 4). As it is a legally binding contract concerning a transaction between *LegalEntities* an *Order* defines the relationships between the partners in a supply chain. An *Order* consists of one or more *OrderItems*. There are various subconcepts of *Order* depending on whether the *Order* is incoming or outgoing and depending on the type of recipient: A final *Customer* issues an *OriginalOrder* addressed to a *Vendor*. This *OriginalOrder* corresponds to the *OrderIncoming* that the *Vendor* receives. Usually this *Vendor* will not produce all the parts needed for the fulfillment of this *OrderIncoming*. Therefore this *OrderIncoming* can trigger an *OrderOutgoing* for the components needed that is addressed to other *Vendors*. It can also trigger a *DeliveryOrderOutgoing* directed to a *Carrier* for the delivery to the final *Customer* or to a *Vendor* if a supplier (another *Vendor*) issues this subtype of *Order*. The *Carrier* receives this as a *DeliveryOrderIncoming*. An *Order* can be identified by its *OrderId*. An *Order* receives an *OrderId* from its issuer as well as from the recipient: An *OrderOutgoing* receives an *OrderOutgoingId*, an *OrderIncoming* an *OrderIncomingId*. For tracking purposes it is necessary to find the corresponding *OrderIncoming/OrderOutgoing* combination. Therefore the *OrderIncoming* also contains the *OrderOutgoingId* of its corresponding *OrderOutgoing*.

Besides the description of a supply chain scenario the *Order* is the point of reference for the tracking data. One important concept derived from the tracking data is the concept of a *Milestone*, that belongs to the status data in specific to the type of time data. A *Milestone* is the unit of the *Effect* of an executed *Activity* and the *CalendarDate* of its achievement. For each *Activity* a *Milestone* is defined. During the fulfillment process, the *Milestones* are realized when the respective *Activities* have been executed. Other types of status data that are incorporated in the ontology but not further explained here refer to

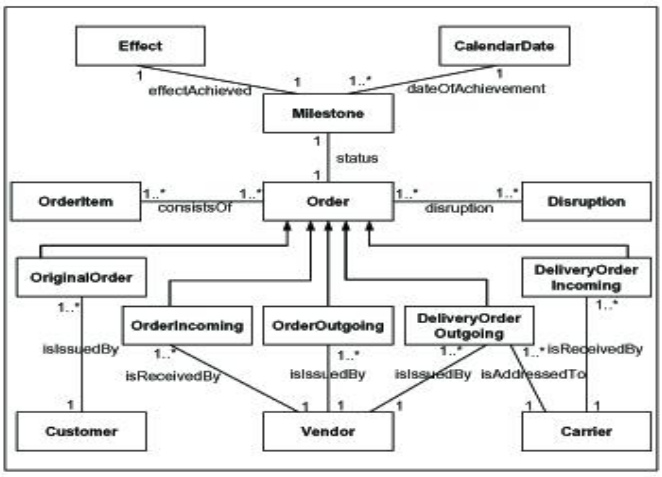


Figure 4: Order Concept.

the group of quality measures that can be derived from measures of time and of quantities of an order (e.g. the relation of delivered to ordered quantity). Apart from status data, so called decision data has been modeled in the ontology. During the fulfillment process of an *Order*, *Disruptions* may occur. These can be *OrderProcessingDisruptions* such as an *OrderLost*, *ProductionDisruptions* such as *MaterialNotAvailable* or a *MachineFailure*, or *DeliveryDisruptions* such as a *TrafficJam*.

The concept of *Time* plays an important role for tracking *Orders* along the supply chain. However, it is not specific to this environment. The Enterprise Ontology uses a *Time* concept imported from Allen's work [5]. This concept is also used for the supply chain monitoring ontology. *Activities* are performed over a *TimeRange* that is comprised of two *TimePoints*: a start and an end date, represented by *CalendarDates*.

4 Conclusion

Irregularities and disruptions in fulfillment processes have a major influence on supply chain performance. It is necessary to optimize the processes of gathering and communicating information related to such events. An agent-based concept presented for supply chain monitoring allows to speed up these processes and to focus on tracking critical orders. To enable the communication of tracking information between the software agents, an ontology has been developed that comprises relevant tracking information and important concepts of a supply chain scenario. The formalized model enables the efficient reuse of the ontology in different applications. As a result complex, dialogs with other supply chain partners agents emerge using the ontology as the necessary model to specify

the content of the agent communication and to interpret the received message to determine appropriate actions. Currently, a first prototype of the supply chain monitoring framework is being implemented in cooperation with a business partner, and the ontology is used to implement an advanced prototype of the discourse agent with reasoning capabilities.

References

[1] D. Fensel. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Berlin, 2001.

[2] V. Haarslev and R. Möller. Description of the racer system and its applications. In *International Workshop on Description Logics (DL-2001)*, Stanford, August 2001.

[3] C.W. Holsapple and K.D. Joshi. A collaborative approach to ontology design. *Communications of the ACM*, 45(2), February 2002.

[4] I. Hurrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, F. Goble C., van Harmelen, M. Klein, S. Staab, R. Studer, and E. Motta. The ontology inference layer oil. http://www.ontoknowledge.org/oil/papers.shtml.

[5] Allen. J.F. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[6] A. Stein, W. Krieger, A. Pflaum, and H. Dräger. Sendungsverfolgung zwischen Marketinginstrument und Produktionsunterstützungstool. In *GVB Schriftenreihe*, volume 40, pages 20–37. 1998.

[7] T. Teufel, J. Röhricht, and P. Willems. *SAP-Prozesse: Planung, Beschaffung und Produktion*. Addison-Wesley, München, 2000.

[8] M. Uschold and M. King. Towards a methodology for building ontologies. Workshop on Basic Ontological Issues in Knowledge Sharing (IJCAI '95), 1995.

[9] M. Uschold, M. King, S. Moralee, and Y. Zorgios. The enterprise ontology. http://www.aiai.ed.ac.uk/ entprise/enterprise/ontology.html#papers, 1997.

[10] R. Zimmermann, R. Butscher, and F. Bodendorf. An ontology for agent-based supply chain monitoring. accepted for Workshop Agent Technologies in Logistics at ECAI 2002, 2002.

Approximation from a Description Logic with disjunction to another without disjunction

Chan Le Duc, Nhan Le Thanh
Laboratoire I3S, Université de Nice-Sophia Antipolis
Email: cleduc@i3s.unice.fr, Nhan.Le-Thanh@unice.fr

Abstract

Even despite the studies carried out concerning Description Logics with conjunction, the structural approach for standard and non-standard inferences in Description Logics with disjunction remains a subject relatively unexplored. The first studies of approximation inference were made by S. Brandt, R.Küsters and A.Turhan. The authors have proposed a *double exponential* algorithm for computing the upper approximation from an $\mathcal{ALC}$-concept description to an $\mathcal{ALE}$-concept description ($\mathcal{ALC}$-$\mathcal{ALE}$). In many application areas, this double exponential algorithm is largely unsatisfactory. Firstly, this paper aims to study subsumption in $\mathcal{ALU}$ ($\mathcal{ELU}$) language. An approximation algorithm from an $\mathcal{ALU}$-concept description to $\mathcal{FL}_{\neg}$-concept description can be deduced directly from this subsumption. Secondly, this paper proposes an *exponential* algorithm for computing the upper approximation $\mathcal{ALC}$-$\mathcal{ALE}$. The technique used for this algorithm is based on the possibility of distributing the approximation computation over disjuncts and conjuncts.

1 Introduction

In electronic commerce, a transaction consists of document interchanges between partners [11]. Composition and interpretation of these documents require a vocabulary for each partner as well as a mechanism capable of translating one vocabulary into another. However, electronic commerce is a heterogeneous domain and consists of many different subdomains. Each subdomain uses its own vocabulary for internal document interchanges. At present,

document interchange between subdomains in semantics tranparency poses a constant challenge. Fortunately, however, recent use of ontologies for design of such vocabularies can be considered as a solution for the problem. Each subdomain has its own vocabulary, all of which derive from a common ontology. These derived ontologies are probably designed with the use of different DL languages according to required expressiveness. In such a system, one of the essential questions is how to interprete a non-defined term found in a received document.
This study addresses the problem which arises when automatically translating from a derived ontology in an expressive DL into another ontology (sharing the same base concepts) in less expressive DL. The interpreting mechanism between derived ontologies, therefore, requires an efficient algorithm which would allow us to exactely or approximately compute a L_s-concept description (source) into L_d-concept description (destination). In the case where the precise interpretation does not exist, the approximative computation would allow us to propose a "nearest" L_d-concept description w.r.t subsumption relation.
We begin by investigating the subsumption problem of $\mathcal{ALU}$ in Section 3. In attempting to extend the automata theory method mentionned in [1] for $\mathcal{ALU}$ but obtained result was not as satisfactory as first hope: we could not represent an $\mathcal{ALU}$-concept description as a automaton. The result archived was instead a pushdown automaton. This allows us, however, to interprete the semantics of an $\mathcal{ALU}$-concept description as being tree-like in structure. This section is then terminated by an algorithm for computing approximation $\mathcal{ALU}$-$\mathcal{FL}_\neg$. This approximation is based on subsumption characterization in $\mathcal{ALU}$.
In section 4, we show a interesting property of *lcs* (least common subsumer) in $\mathcal{ALE}$ which states that for *propagated* $\mathcal{ALE}$ concept descriptions, the common conjuncts are also conjuncts of *lcs*. Nevertheless, before computing of $\mathcal{ALC}$-$\mathcal{ALE}$ approximation, necessary transformations must be carried out on $\mathcal{ALC}$-concept description. These transformations will allow us to distribute the approximation computation in $\mathcal{ALE}$ of an $\mathcal{ALC}$-concept description over its disjuncts and conjuncts. This technique is one of crucial points that will avoid the double exponential complexity in the approximation algorithm.

2 Description Logics

Concept descriptions are built from concept constructors, a set N_C of concept names and a set N_R of role names. The semantics of a concept description in some language is defined with the aide of the following interpretation $I=(\Delta, \cdot^I)$. The following table gives a summary of languages that are used in

this paper. Note that the largest language used in this paper is $\mathcal{ALC}$. This language allows concept descriptions to be negated whereas in $\mathcal{ALE}$ negation is only allowed in front of concept names.
Subsumption inference used in this paper is defined as follows : the concept description C is subsumed by the description D, written $C \sqsubseteq D$, iff $C^I \subseteq D^I$ holds for all interpretation $\mathcal{I}$.

Syntax	Semantics	$\mathcal{FL}_\neg$	$\mathcal{ALU}$	$\mathcal{ELU}$	$\mathcal{ALE}$	$\mathcal{ALC}$
$\top$	Δ	x	x	x	x	x
$\bot$	$\emptyset$	x	x	x	x	x
$C \sqcap D$	$C^I \cap D^I$	x	x	x	x	x
$\forall r.C$	$\{x \in \Delta \mid \forall y{:}(x,y) \in r^I \rightarrow y \in C^I\}$	x	x		x	x
$\exists r.C$	$\{x \in \Delta \mid \exists y{:}(x,y) \in r^I \wedge y \in C^I\}$			x	x	x
$\neg A, A \in N_C$	$\Delta \setminus A^I$	x	x	x	x	x
$C \sqcup D$	$C^I \cup D^I$		x	x		x
$\neg C$	$\Delta \setminus C^I$					x

3 Characterization of subsumption in $\mathcal{ALU}(\mathcal{ELU})$ and $\mathcal{ALU}$-$\mathcal{FL}_\neg$ approximation

We will begin by investigating the problem of subsumption in $\mathcal{ALU}(\mathcal{ELU})$. Firstly, let us provide the following definition in order to clarify the notion of approximation.

Definition 3.1. *Let L_s and L_d be two DLs, and let C be a L_s-concept description and D be a L_d– concept description. Then, D is called upper (lower) L_d-approximation of C (min(C), max(C) for short) if*
i) $C \sqsubseteq D$ ($D \sqsubseteq C$) and
ii) $C \sqsubseteq D'$ ($D' \sqsubseteq C$), $D' \sqsubseteq D$ ($D \sqsubseteq D'$) implies $D' \equiv D$ for all L_d-concept descriptions D'

Remarks: Let L_s=$\mathcal{ALC}$, L_d=$\mathcal{ALE}$. If there exists $min(C)$ then it is unique. Indeed, if D_1= $min(C)$ and D_2= $min(C)$ then $C \sqsubseteq D_1 \sqcap D_2 \sqsubseteq D_1$ and $C \sqsubseteq D_1 \sqcap D_2 \sqsubseteq D_2$. It means that $D_1 \equiv D_2$. But there may be many $max(C)$. For example, $C = \exists r.P \sqcup \forall r.Q$, $max'(C) = \exists r.P$, $max"(C) = \forall r.Q$.

Now, we will provide the following normalization definition allowing us to postpone processing conjunctions.

Definition 3.2. *A term is called $\forall$-normal iff it is disjunction of terms*

of the form : $\forall r.(A \sqcup B)$ *where A,B are also* $\forall$*-normal terms and none of the following rules can be applied at any position in the term :*

$$\begin{aligned} P \sqcup \neg P &\rightarrow \top \\ E \sqcup \bot &\rightarrow E \\ \forall r.\top &\rightarrow \top \\ E \sqcup \top &\rightarrow \top \end{aligned}$$

Remarks: i) $\forall$-*normal* terms do not contain conjunction.
ii) Any $\mathcal{ALU}$-concept description C can be represented as conjunctions of $\forall$-*normal* terms. Indeed this transformation can be made providing that two following properties : $(A \sqcap B) \sqcup C = (A \sqcup C) \sqcap (B \sqcup C)$ and $\forall w.(A \sqcap B) = \forall w.A \sqcap \forall w.B$. This transformation can take an exponential time in the size of C.
iii) Replacing $\forall$ and $\sqcup$ with respectively $\exists$ and $\sqcap$ in the definition, we will obtain the definition of $\exists$-*normal* term. In the same way, any $\mathcal{ELU}$-concept description can be represented as disjunctions of $\exists$-*normal* terms.
iv) We denote $Q_i \in \{P_i | P_i \in N_C\} \cup \{\neg P_i | P_i \in N_C\}, r_i \in N_R$.

Definition 3.3. *A is a* $\forall$*-normal term. The language of A, called L(A), is defined recursively as follows:*

$$\begin{aligned} &(I) \quad L(Q) = \{(Q).\varepsilon\} = \{(Q)\}; L(Q_1 \sqcup Q_2) = \{(Q_1, Q_2).\varepsilon\} = \{(Q_1, Q_2)\} \\ &(II) \quad L(A_1 \sqcup A_2) = \{L(A_1) \cup L(A_2)\} \\ &(III) \quad L(\forall r_1.(A_1 \sqcup A_2))) = \{(L(A_1) \cup L(A_2)).r_1\} \end{aligned}$$

Remarks: i) $L(A)$ is equivalent to a pushdown automaton. ii) For all words $w \in L(A)$, w can be written as $w = L.u$ where L is language of a $\forall$-normal term and $u = r_1 \ldots r_m, r_i \in N_R$. iii) Set of paths of a word $w \in L(A)$, called $C(w)$, is defined as follows: $C((Q).\varepsilon) = \{Q\}$; $C((L(A_1) \cup L(A_2)).r) = \{C(L(A_1)).r \cup C(L(A_2)).r\}$. Since set of paths $C(w)$ does not contain structural informations, a word $w \in L(A)$ is not equivalent to the set of paths $C(w)$.
We now can characterize the semantics of a $\forall$-normal term using a tree which is defined as follows.

Definition 3.4. (Construction of normal term tree)
A is a $\forall$-*normal* term. The tree of the term A, T_A is defined as follows:

1. The root n_0 of T_A is created.

2. If $A = Q$ or $A = Q_1 \sqcup Q_2$, respectively a node (Q) or (Q_1, Q_2) is created. The root n_0 is replaced by this node which becomes the root of T_A. If $A = \forall r.(Q_1 \sqcup Q_2)$, a node $n_1 = (Q_1, Q_2)$ is created. The tree T_A is constructed by adding an edge r in order to connect the node n_1 with n_0.

3. If $A{=}(A_1 \sqcup Q_1 \sqcup Q_2)$ and assume that the tree T_{A_1} of A_1 is built. Firstly, the root of T_{A_1} is replaced by the root node n_0. And then a node (Q_1, Q_2) which unifies all primitive concepts at top-level is created. This node is connected with the root node n_0 by an edge ε. If $A{=}A_1 \sqcup A_2$ and assume that the trees T_{A_1}, T_{A_2} are built. Two roots of these trees are unified in a unique node which is replaced by the root node n_0.

4. If $A = \forall r_1.(A_1 \sqcup A_2))$ and assume that two subtrees T_{A_1} and T_{A_2} are built. Firstly, two roots of the trees T_{A_1} and T_{A_2} are unified in a unique node n_1. And then the tree T_A is constructed by adding an edge r_1 in order to connect n_0 with n_1.

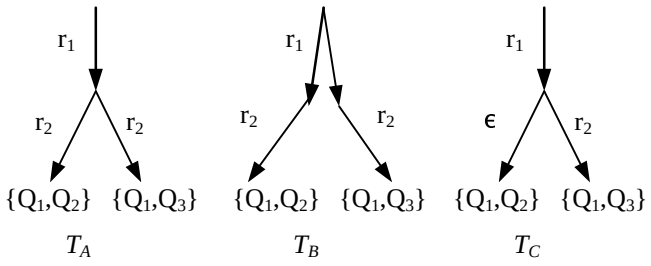


Figure 1: Trees T_X of $\forall$-normal terms

Remarks : i) In the construction of tree T_A, each leaf of tree T_A will be a set $\{Q_i\}$ i.e primitive concepts in $\{(Q_1), \ldots, (Q_m)\}.r$ are unified to a leaf $(Q_1, \ldots, Q_m)$.
ii) For the language $\mathrm{L}(A)$, the tree T_A which is constructed as in *Definition 3.4* is unique if the order of subtrees is not important.
iii) We denote $w \in C(A)$ as being a path from the root to a leaf of tree T_A and $P_A(w) = \{Q_i | Q_i$ belongs to the leaf $\{Q_i\}$of $w\}$. We write $W(A, Q)$ for set of paths of A terminated at Q in T_A.

Examples 3.1. (Fig. 1)
$A = \forall u_1.(\forall u_2.(P_1 \sqcup P_2) \sqcup (\forall u_2.(P_1 \sqcup P_3))), L(A)=\{((P_1, P_2).u_2, (P_1, P_3).u_2).u_1\}$,
$C(A)=\{(P_1, P_2).u_2.u_1, (P_1, P_3).u_2.u_1\}$.
$B = \forall u_1.u_2.(P_1 \sqcup P_2) \sqcup \forall u_1.u_2.(P_1 \sqcup P_3)$ $L(B)=\{((P_1, P_2).u_2.u_1, (P_1, P_3).u_2.u_1\}$,
$C(B)=\{((P_1, P_2).u_2.u_1, (P_1, P_3).u_2.u_1\}$.
$C = \forall u_1.(P_1 \sqcup P_2 \sqcup (\forall u_2.(P_1 \sqcup P_3))$ $L(C)=\{((P_1, P_2).u_1, (P_1, P_3).u_2.u_1\}$,
$C(C)=\{((P_1, P_2).u_1, (P_1, P_3).u_2.u_1\}$.

Proposition 3.1. *Let T be an acyclic $\mathcal{ALU}$-TBox . Let I be a model of T. Let A be concept name occurring in T and be a $\forall$-normal term. For any $d \in \Delta^I$ (Δ^I=dom(I)), we can build a tree T_A^d w.r.t T_A i.e d is root of*

the tree and there exist individuals e_i corresponding to the nodes and leaves n_k of T_A and each edge $(e_i, e_j) = u_{ij}$ of T_A^d corresponds uniquely to an edge $(n_i, n_j) = u_{ij}$ of T_A. We have:

$d \in A^I$ *iff*

for all trees T_A^d, there exists a path w of the tree T_A such that for all $e \in \Delta^I$: $(d,e) \in w^I \Rightarrow e \in Q_1^I \vee \ldots \vee e \in Q_m^I$ where $Q_i \in P_A(w)$.

A proof of this proposition can be found in [10]. Note that the semantics of an $\exists$-normal term in $\mathcal{ELU}$ can be similarly defined as follow :

Proposition 3.1.1. *Let T be an acyclic $\mathcal{ELU}$-TBox . Let I be a model of T. Let A be a concept name occurring in T and be a $\exists$-normal term. For any $d \in \Delta^I$ (Δ^I = dom(I)), we can build a tree T_A^d w.r.t T_A i.e d is root of the tree and there exist individuals e_i corresponding to the nodes and leaves n_k of T_A and each edge $(e_i,\ e_j) = u_{ij}$ of T_A^d corresponds uniquely to an edge $(n_i,\ n_j) = u_{ij}$ of T_A. We have:*

$d \in A^I$ *iff*

there exists a tree T_A^d such that for all paths w of the tree T_A there exists $e \in \Delta^I$: $(d, e) \in w^I \wedge e \in Q_1^I \wedge \ldots \wedge e \in Q_m^I$ where $Q_i \in P_A(w)$.

Therefore, the results obtained for $\mathcal{ALU}$ can be extended to $\mathcal{ELU}$.
There are at hand the notion of language of a $\forall$-normal term, we now can define an order relation on set of languages as the following.

Definition 3.5. *Let $L(A)$ and $L(B)$ be languages of $\forall$-normal terms.*

(I) *For all $L(A)$: $L(\bot) \preceq L(A)$, $L(A) \preceq L(\top)$.*
(II) *$L(Q_m \sqcup \ldots \sqcup Q_n) \preceq L(Q_p \sqcup \ldots \sqcup Q_q)$ iff $\{Q_m, \ldots, Q_n\} \subseteq \{Q_p, \ldots, Q_q\}$.*
(III) *$L(A) \preceq L(B)$ iff for all words m=$L_m.u \in L(A)$ there exists a word $n = L_n.v \in L(B)$ such that $u = v$ and $L_m \preceq L_n$.*
(IV) *$T_A \preceq T_B$ iff $L(A) \preceq L(B)$.*

From the partial relation that is defined on set of languages in *Definition 3.5*, we can characterize subsumption relation on set of $\forall$-*normal* terms.

Proposition 3.2. *Let T be an acyclic $\mathcal{ALU}$-TBox . Let I be a model of T. Let A_1 and A_2 be $\forall$-normal terms occurring in T, we have:*

$A_1 \sqsubseteq A_2$ iff $L(A_1) \preceq L(A_2)$.

A proof of *Proposition 3.2* can be found in [10]. In example 3.1, from Proposition 3.2 we have $B \sqsubseteq A$.

We now extend these results for the language $\mathcal{ALU}$. Firstly, we define language $L(C)$ and tree T_C of an $\mathcal{ALU}$-concept description C. This definition also allows us to determine $\forall$-normal form of an $\mathcal{ALU}$-concept description C.

Definition 3.6. *Let C be an $\mathcal{ALU}$-concept description.*

(I) *The $\forall$-normal form of C : $C = C_1 \sqcap \ldots \sqcap C_m$ where C_i are $\forall$-normal terms and all $\top, \bot$ are eliminated at top-level by the following rules : $P \sqcap \neg P \rightarrow \bot$, $E \sqcap \top \rightarrow E$ and $E \sqcap \bot \rightarrow \bot$.*
(II) $L(C_1 \sqcap \ldots \sqcap C_m) = \{L(C_1), \ldots, L(C_m)\}$.
(III) $T_{(C_1 \sqcap \ldots \sqcap C_m)} = \{T_{C_1}, \ldots, T_{C_m}\}$.

Remarks : i) According to the *Remarks* of Definition 3.2, an $\mathcal{ALU}$-concept description C can be always transformed into $\forall$-normal form. This transformation can take an exponential time in the size of C. ii)The semantics of C are naturally translated into the semantics of the set of trees T_{C_i}.

Theorem 3.1. *Let T be a acyclic $\mathcal{ALU} - TBox$. Let I be a model of T. Let A_1 and A_2 be concept descriptions occurring in T where $A_1 = A_{11} \sqcap \ldots \sqcap A_{1u}$, $A_2 = A_{21} \sqcap \ldots \sqcap A_{2v}$ are $\forall$-normal forms. We denote: $L(A_1) = \{L(A_{11}), \ldots, L(A_{1u})\}$, $L(A_2) = \{L(A_{21}), \ldots, L(A_{2v})\}$. We have :*

$A_1 \sqsubseteq A_2$ *iff for all $L(A_{2j}) \in L(A_2)$, there exists $L(A_{1i}) \in L(A_1)$ such that $L(A_{1i}) \preceq L(A_{2j})$.*

A proof of *Theorem 3.1* can be found in [10]. Our problem in this section is, however, to find a $\mathcal{FL}_\neg$-approximation D of an $\mathcal{ALU}$-concept description C. *Corollary 3.1* will show that the subsumption characterization in language $\mathcal{ALU}$ is an extension of method using automata theory mentionned in [1]. This relationship allows us to compute the approximation $\mathcal{ALU}-\mathcal{FL}_\neg$.

Corollary 3.1. *Let C be L_s-concept description where $L_s = \mathcal{ALU}$ and $L_d = \mathcal{FL}_\neg$. There exists an upper approximation D of C in L_d and the computing of the approximation can take at most an exponential time in the size of C.*

Proof: We denote $\mathrm{T}_C = \{\mathrm{T}_{C_1}, \ldots, \mathrm{T}_{C_m}\}$ as a set of trees where C is considered as a conjunction of C_i terms and T_{C_i} corresponds to C_i. According to [1], each concept description D in $\mathcal{FL}_\neg$ is characterized by the words of $\mathrm{L}(D, Q_i)$ which can be considered as trees terminated by Q_i. Hence, we can build T_D from words of $\mathrm{L}(D, Q_i)$ for all Q_i. Consequently, we obtain a necessary and sufficient condition for the expressiveness of C in L_d from *Theorem 3.1* where it states that each T_{C_i} must be reduced to a path.
Existence and computing of the upper approximation D of C in L_d : $C \sqsubseteq D$.

Assume that there exist T_{C_k} such that T_{C_k} are paths. We denote the corresponding concept descriptions : $C_{k_1}, \ldots, C_{k_n}$. In this case, from *Theorem 3.1* the upper approximation D is built as a conjunction of C_{k_i}: $D = min^{\mathcal{ALU}}_{\mathcal{FL}_\neg} = \prod_{i=1}^{n} C_{k_i}$. If there does not exist T_{C_k} where T_{C_k} is a path, then $D = min^{\mathcal{ALU}}_{\mathcal{FL}_\neg} = \top$. The $\forall$-normalization of C needs at most an exponential time in the size of C. The research of path trees T_{C_k} takes also at most an exponential time in the size of C. Hence, whole computation can be carried out in exponential time in the size of C. ■

An algorithm can be built directly from *Corollary 3.1* which allows us to compute the upper approximation D in $\mathcal{FL}_\neg$ from a concept description C in $\mathcal{ALU}$ up to equivalence.

Example 3.2.
i) $min_{\mathcal{FL}_\neg}(\mathrm{A}) = \top$; $min_{\mathcal{FL}_\neg}(A \sqcap \forall r.P_1 \sqcap \forall r.P_2) = \forall r.(P_1 \sqcap P_2)$ where A is defined in Example 3.1.

ii) Let $C = \forall r.P_1 \sqcap \forall r.(P_1 \sqcup \neg P_1) \sqcap \forall r.(P_2 \sqcap (P_3 \sqcup P_4))$.
Normalizing: $C = \forall r.P_1 \sqcap \forall r.\top \sqcap \forall r.P_2 \sqcap \forall r.(P_3 \sqcup P_4) = \forall r.P_1 \sqcap \forall r.P_2 \sqcap \forall r.(P_3 \sqcup P_4)$.
Finding path trees : $\forall r.P_1, \forall r.P_2$.
$min^{\mathcal{ALU}}_{\mathcal{FL}_\neg}(C) = \forall r.P_1 \sqcap \forall r.P_2$.

4 $\mathcal{ALU}$-$\mathcal{ALE}$ ($\mathcal{ELU}$-$\mathcal{ALE}$) and $\mathcal{ALC}$-$\mathcal{ALE}$ approximations

In this section, we propose an approach for computating the $\mathcal{ALE}$-approximation of a concept description containing disjunction. This approach is based on three events : i) computing the normal forms w.r.t conjunction and disjunction. ii) computing *lcs* (*least common subsumer*) of disjuncts on top-level. iii) propagating value restrictions on existential restrictions of conjuncts on top-level.
For the sake of simplicity, we assume that the set N_R of role names is $\{r\}$. However, all obtained results in this section can be generalized to arbitrary sets of role names.

Definition 4.1. (the following notations are used in [3]) *C is an $\mathcal{ALC}$-concept description* (c for conjunction, d for disjunction).

- *$PRIM^c(C)$ or $PRIM(C)$ ($PRIM^d(C)$) denotes the set of all (negated) concept names and the bottom (top) concept occurring on the top-level conjunction (disjunction) of C.*

- *$VAL^c_r(C)$ or $VAL_r(C)$ ($VAL^d_r(C)$) is conjunction (a set) of all C' occurring in value restrictions of the form $\forall r.C$' on top-level of C. If there is no value restriction on top-level of C then $VAL^c_r(C) = \top$.*

- *$EX^d_r(C)$ ($EX^c_r(C)$ or $EX_r(C)$) is set (disjunction) of all C' occurring in existential restrictions of the form $\exists r.C$' on top-level of C.*

- *The d-normal form of C (d-normal(C))*
  $C = C_1 \sqcup \ldots \sqcup C_m$ where
  $C_i = \sqcap_{A \in PRIM(Ci)} A \sqcap \prod_{C' \in EXr(Ci)} \exists r.C' \sqcap \forall r.VAL^c_r(C_i)$,
  $\bot \sqsubset C_i$, C' and $VAL^c_r(C_i)$ are in d-normal forms.

- *The c-normal form of C (c-normal(C))*
  $C = C_1 \sqcap \ldots \sqcap C_n$ where
  $C_i = \sqcup_{A \in PRIM(Di)} A \sqcup \exists r.EX^d_r(C_i) \sqcup \bigsqcup_{C' \in VALr(Di)} \forall r.C'$
  $C_i \sqsubset \top$, C'and $EX^d_r(C_i)$ are in c-normal forms.

Remarks: i) If C is an $\mathcal{ALC}$-concept description, C can be turned into an equivalent concept description of the *d-normal* form or an equivalent concept description of the *c-normal* form.
ii) *c-normal* form and *d-normal* form of an $\mathcal{ALC}$-concept description can be exponentially larger than C itself.
We now define *propagated normal form* of an $\mathcal{ALC}$-concept description. This is an extension of the *normal form* mentionned in [5] for $\mathcal{ALE}$-concept description. The propagated normal form of *c-normal* form of an $\mathcal{ALC}$-concept description allows us to compute the approximation $\mathcal{ALC}$-$\mathcal{ALE}$ as a conjunction of approximations on smaller terms.

Definition 4.2. *Let C be an $\mathcal{ALC}$-concept description. C is in propagated normal if none of the following rules can be applied at top-level in C_i :*

$$\begin{array}{rcl} P \sqcap \neg P & \rightarrow & \bot \\ E \sqcap \bot & \rightarrow & \bot \\ \exists r.\bot & \rightarrow & \bot \\ \forall r.\top & \rightarrow & \top \\ E \sqcap \top & \rightarrow & \top \\ \forall r.E \sqcap \forall r.F & \rightarrow & \forall r.(E \sqcap F) \\ \forall r.E \sqcap \exists r.F & \rightarrow & \forall r.E \sqcap \exists r.(E \sqcap F) \end{array}$$

where C_i are terms occurring in d-normal form of $C = C_1 \sqcup \ldots \sqcup C_m$.
Remarks. i) Propogated normal transformation increases polynomially the size of the d-normal form of C.

Definition 4.3. *Let $C_1, \ldots, C_n$ be the L_- concept descriptions. The L_- concept description C is the least common subsumer (lcs) of $C_1, \ldots, C_n$ ($C = lcs(C_1,$*

...,C_n) for short) iff i) $C_i \sqsubseteq C$ for all i=1..n ii) $C_i \sqsubseteq C'$ for all i=1..n that implies $C \sqsubseteq C'$.

The main idea of the efficient algorithm is based on the possibility of distributing approximation computation over disjuncts and conjuncts. However, the distribution only holds for disjuncts owing to the application of *lcs* on the distribution. A further normalization is required for conjunctions before the distribution. In fact, the propagated normal form allows the distribution to hold for conjunction. This property is guaranteed by the following proposition.

Proposition 4.1.

1. *If $A = A_1 \sqcap C$ and $B = B_1 \sqcap C$ are propagated $\mathcal{ALE}$-concept descriptions i.e the rules in Definition 4.2 are applied to $\mathcal{ALE}$-concept descriptions, we have : lcs(A, B) = C⊓ lcs(A_1, B_1).*

2. *If $C = C_1 \sqcup C_2$ where C is an $\mathcal{ALC}$-concept description and $\bot \sqsubset C_1, C_2$ then, $min_{\mathcal{ALE}}(C) = lcs(min_{\mathcal{ALE}}(C_1), min_{\mathcal{ALE}}(C_2))$.*

3. *If $C = C_1 \sqcap C_2$ is a propagated normal $\mathcal{ALC}$-concept description, we have : $min_{\mathcal{ALE}}(C) = min_{\mathcal{ALE}}(C_1) \sqcap min_{\mathcal{ALE}}(C_2)$.*

A proof of *Proposition 4.1* can be found in [10]. Naturally, we immediately have an algorithm computing approximation $\mathcal{ALU}$-$\mathcal{ALE}$:

Input: $\mathcal{ALU}$-concept description C Output: $min_{\mathcal{ALE}}(C)$

- If $C = \forall r.C_1$ $(C = \exists r..C_1)$, $min_{\mathcal{ALE}}(C) = \forall r.(min_{\mathcal{ALE}}(C_1))$ $(min_{\mathcal{ALE}}(C) = \exists r.(min_{\mathcal{ALE}}(C_1))$
- If C can be written in $C = C_1 \sqcup C_2$ where $\bot \sqsubset C_1, C_2$, $min_{\mathcal{ALE}}(C) = lcs(min_{\mathcal{ALE}}(C_1), min_{\mathcal{ALE}}(C_2))$
- If $C = C_1 \sqcap C_2$ where $C_1, C_2 \sqsubset \top$, $min_{\mathcal{ALE}}(C) = min_{\mathcal{ALE}}(C_1) \sqcap min_{\mathcal{ALE}}(C_2)$

Figure 2: Algorithm for $\mathcal{ALU}(\mathcal{ELU})$-$\mathcal{ALE}$ approximation

Theorem 4.1. *Let C be an $\mathcal{ALU}$ $(\mathcal{ELU})$-concept description.*

1. *If $C \equiv \bot$ or $C \equiv \top$ then $min_{\mathcal{ALE}}(C) = \bot$ or $min_{\mathcal{ALE}}(C) = \top$.*

2. *If* $C = C_1 \sqcup C_2$ *where* $\bot \sqsubset C_1$, C_2 *then,*
   $min_{\mathcal{ALE}}(C) = lcs(min_{\mathcal{ALE}}(C_1),\ min_{\mathcal{ALE}}(C_2))$.

3. *If* $C = C_1 \sqcap C_2$ *where* C_1, $C_2 \sqsubset \top$, *then*
   $min_{\mathcal{ALE}}(C) = min_{\mathcal{ALE}}(C_1) \sqcap\ min_{\mathcal{ALE}}(C_2)$.

A proof of *Theorem 4.1* can be found in [10].

Corollary 4.1. *The algorithm* $min^{\mathcal{ALC}}_{\mathcal{ALE}}(C)$ *takes at most an exponential time in the size of* C *where* C *is an* $\mathcal{ALU}(\mathcal{ALE})$*-concept description.*
A priori, the algorithm that is introduced in *Theorem 4.1* can be extended to approximation $\mathcal{ALC}$-$\mathcal{ALE}$. However, $\mathcal{ALC}$ is not a propagated normal language by default. Therefore, so that $\mathcal{ALC}$ becomes propagated normal, further transformations are required. *Theorem 4.2.* shows how this process operates.

Theorem 4.2. *Let* C *be an* $\mathcal{ALC}_{-}$*concept description.*

1. *If* $C \equiv \bot$ *or* $C \equiv \top$ *then* $min_{\mathcal{ALE}}(C) = \bot$ *or* $min_{\mathcal{ALE}}(C) = \top$.

2. *If* $C = C_1 \sqcup C_2$ *where* $\bot \sqsubset C_1$, C_2 *then,*
   $min_{\mathcal{ALE}}(C) = lcs(min_{\mathcal{ALE}}(C_1),\ min_{\mathcal{ALE}}(C_2))$.

3. *If* $C = C_1 \sqcap \ldots \sqcap C_m$ *where* $C_i \sqsubset \top$. *We have* $min_{\mathcal{ALE}}(C) = \bigsqcap_q min_{\mathcal{ALE}}(E_q)$ *where the size of* E_i *is a polynomial in the size of* C *and the number of terms* E_i *is exponential in the size of* C.

A proof of *Theorem 4.2* can be found in [10].

Corollary 4.2. *The algorithm* $min^{\mathcal{ALC}}_{\mathcal{ALE}}(C)$ *takes an exponential time in the size of* C *where* C *is an* $\mathcal{ALC}$*-concept description.*
A proof of *Corollary 4.2* can be found in [10].

Example 4.1.
$C = \forall r.B \sqcap \exists r.A \sqcap\ (\exists r.B \sqcup\ \forall r.B)$.

(Step 3.1) Computing *d-normal* of C : $(\forall r.B \sqcap \exists r.A \sqcap \exists r.B) \sqcup (\forall r.B \sqcap \exists r.A)$
(Step 3.2) Propagating *d-normal* : $(\forall r.B \sqcap \exists r.(A \sqcap B) \sqcap\ \exists r.B) \sqcup (\forall r.B \sqcap$
$\exists r.(A \sqcap B))$
(Step 2.) Computing *c-normal* : $\forall r.B \sqcap\ (\forall r.B \sqcup \exists r.(A \sqcap B)) \sqcap\ (\exists r.(A \sqcap B)$
$\sqcup \forall r.B) \sqcap \exists r.(A \sqcap B) \sqcap\ (\exists r.B \sqcup \forall r.B) \sqcap\ (\exists r.B \sqcup \exists r.(A \sqcap B))$
(Step 3.4) Computing the approximation:
$\min(\mathrm{C}) = \min(\forall r.B) \sqcap\ \min(\forall r.B \sqcup \exists r.(A \sqcap B)) \sqcap$
$\min(\exists r.(A \sqcap B) \sqcup \forall r.B) \sqcap\ \min(\exists r.(A \sqcap B)) \sqcap\ \min(\exists r.B \sqcup \forall r.B)$
$\sqcap\ \min(\exists r.B \sqcup \exists r.(A \sqcap B))$

$= \forall r.B \sqcap \text{lcs}(\forall r.B, \exists r.(A \sqcap B)) \sqcap \text{lcs}(\exists r.(A \sqcap B), \forall r.B) \sqcap$
$\min(\exists r.(A \sqcap B)) \sqcap \text{lcs}(\exists r.B, \forall r.B) \sqcap \min(\exists r.B \sqcup \exists r.(A \sqcap B))$
$= \forall r.B \sqcap \top \sqcap \top \sqcap \exists r.(A \sqcap B) \sqcap \top \sqcap \exists r.B = \forall r.B \sqcap \exists r.(A \sqcap B) \sqcap$
$\exists r.B$ ■

Input: $\mathcal{ALC}$-concept description C
Output: $min_{\mathcal{ALE}}(C)$

1. If $C = \forall r.C_1$ ($C = \exists r.C_1$) , $min_{\mathcal{ALE}}(C) = \forall r.(min_{\mathcal{ALE}}(C_1)$ ($min_{\mathcal{ALE}}(C) = \exists r.(min_{\mathcal{ALE}}(C_1))$

2. If C can be written in $C = C_1 \sqcup ... \sqcup C_n$ where $\bot \sqsubset C_i$
   $min_{\mathcal{ALE}}(C) = lcs(min_{\mathcal{ALE}}(C_1)$,..., $min_{\mathcal{ALE}}(C_n))$

3. If C can be written in $C = C_1 \sqcap ... \sqcap C_n$ where $C_i \sqsubset \top$

   3.1 Computing *d-normal* form of C : *d-normal*(C)= $D_1 \sqcup \ldots \sqcup D_n$

   3.2 For each D_i, replacing $\exists r.C$' with $\exists r.(C' \sqcap VAL_r(D_i))$, where C' $\in EX_r(D_i)$, we obtain *d-normal*$(C) = \bigsqcup_i D_i'$

   3.3 Computing *c-normal*(*d-normal*$(C)) = \bigsqcap_1^q E_i$

   3.4 $min_{\mathcal{ALE}}(C) = \bigsqcap_1^q min_{\mathcal{ALE}}(E_i)$

Figure 3: Algorithm for $\mathcal{ALC}$-$\mathcal{ALE}$ approximation

5 Conclusion

We have proposed a way of characterizing subsumption in $\mathcal{ALU}$. This way can be extended for $\mathcal{ELU}$. From the subsumption characterization, we have proposed an exponential algorithm in the worst case for approximating an $\mathcal{ALU}$-concept description with $\mathcal{FL}_\neg$-concept description. We have also introduced an efficient algorithm for computing upper approximation of an $\mathcal{ALC}$-concept description with $\mathcal{ALE}$-concept description. This algorithm runs in exponential time in the size of the $\mathcal{ALC}$-concept description. At present, we know that only structural approach allows us to develop non-standard inference services. An interesting question is how to use structural approach for subsumption in $\mathcal{ALC}$ in order to extend approximation inference to expressive languages. On the other hand, in electronic commerce the interpretation of received terms depends on context informations. It remains to be seen how we may integrate context informations into ontologies. Our work in the future aims to address these questions.

References

[1] Franz Baader. Using automata theory for characterizing the semantics of terminological cycles. *Annals of Mathematics and Artificial Intelligence*, 1996.

[2] Franz Baader, Ralf Küsters, and Ralf Molitor. Rewriting Concepts Using Terminologies. *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning* (KR2000).

[3] Sebastian Brandt, Ralf Küsters, Anni-Yasmin Turhan. Approximation and Difference in Description Logics. *Proceedings of the Eight International Conference on Principles of Knowledge Representation and Reasoning* (KR2002).

[4] Ralf Küsters. Non-standard Inferences in Description Logics. *Thesis, Springer* (2001)

[5] Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumer in description logics with existential restrictions. *Proceedings of the 17th International Joint Conference Artificial Intelligence* (IJCAI 1999)

[6] Catriel Beeri, Alon Y. Levy and Marie-Christine Rousset. Rewriting Queries Using Views in Description Logics. *Proceedings of the 16^{th} ACM Symposium on Principles of Database Systems (PODS-97), Tucson* (May 1997)

[7] F.M Donini, M. Lenzerini, D. Nardi. Reasoning in Description Logics. *CSLI Publications* (1997)

[8] M. Schmidt-Schauss and D.Smolka. Attributive Concept Descriptions with Complements. *Artificial Intelligence* (1991)

[9] Bernhard Nebel. Reasoning and revision in hybrid representation systems. *Thesis, Springer-Verlag* (1990)

[10] Chan Le Duc, Nhan Le Thanh. Approximation from a Description Logic with disjunction to another without disjunction. *Technical Report 2002.* See http://www.i3s.unice.fr/~cleduc.

[11] Chan Le Duc, Nhan Le Thanh. Problématique de l'ebXML et Logique de Description. *Technical Report 2002.* See http://www.i3s.unice.fr/~cleduc.

Corega Tabs: Mapping Semantics onto Pragmatics

Bernd Ludwig, Kerstin Bücher, and Günther Görz
Computer Science Institute and FORWISS, Erlangen, Germany
eMail: bdludwig@immd5-wv.uni-erlangen.de

Abstract

In this paper, we outline our approach to mapping domain independent semantic representations to extensional statement specific to a given application. Fur this purpose, we exploit the equivalence between Discourse Representation Structures limited to the expressiveness of ALC and ABoxes for validating DRS with respect to a given knowledge base and explain how task independent semantics representations can be specialized to a domain specific meaning. The paper concludes with some performance results and remarks on desirable features of DL reasoners not implemented up to now.

1 DL Models of Applications

Applications are characterized by a DL terminology which models the concepts used for making propositions about application situations. In [BLG02] it is explained, how domain independent semantic representations for a certain class of natural language phrases can be composed relying on EuroWordNet (EWN) [Vos98] as a linguistic ontology. The remaining question is, how these representations can be mapped onto ABoxes only containing propositions in a application specific ontology (sec. 2). As a prerequisite, we address the issue of how those two ontologies can be linked to form a modular DL knowledge base composed of several smaller parts for covering special purpose ontologies (sec. 4).

Basically, the knowledge base is composed of two parts. The EWN ontology encodes the linguistic meaning of words determined on an empirical basis, whereas the Standard Upper Ontology (SUMO) [NP01] is used as a generic base model for concepts of the application domain. See section 2 for a description of how we use the knowledge base. We present our experiments with Racer in section 3. The modular composition of T-Boxes is topic of section 4, and section 5 gives an overview about further requirements of A-Box reasoning.

Thematic Role	Inverse	Thematic Role	Inverse
has-mero-part	has-holo-part	be-in-state-of	state-of
has-mero-portion	has-holo-portion	in-manner	manner-of
has-mero-location	has-holo-location	involved-source	
has-mero-member	has-holo-member	involved-result	
has-mero-madeof	has-holo-madeof	involved-agent	
involved-location	role-location	involved-target	
is-subevent-of	has-subevent		
causes	is-caused-by		

Table 1: Thematic Roles in EuroWordNet

2 Mapping Semantics onto Pragmatics

This section discusses the mapping from DRS composed during parsing [BLG02] to ABoxes representing propositions on the current application situation. This means ABoxes have to be consistent with respect to a given TBox. The rationale for using DL for constructing natural language semantics is the possibility to eliminate hypotheses constructed by the parser if the corresponding ABoxes are inconsistent. Any other hypothesis which passes this test is called *admissible*.

2.1 EuroWordNet

In order to ease the adaptability of the dialogue system to different domains and to reflect general and domain independent usage of language from that of a specific application the semantics of chunks is expressed in terms of concept expressions taken from the EuroWordNet terminology. EuroWordNet has been developed on the basis of the WordNet semantic net which – in version 1.5 – encodes the meaning of about 80.000 nouns, 60.000 verbs and 16.000 adjectives and adverbs. Beyond being a pure taxonomy of semantic types, EuroWordNet can be used to define complex concepts for complements verbs and nouns may take in the German language. In a DL approach to define them, relations between primitive concepts are expressed by roles whereas several different complements for a lexical base form are stated using conjunction of concepts. The linguistic notion of synonymy can be implemented in a DL knowledge base via concept equivalence, antonymy by the use of the negation operator. Disjunction is the tool to state alternative usage of language – for example of different words for the same semantic notion. An example of such a concept description is given by the following definition:

$$\texttt{GetOn1BeOn1} \sqsubseteq \texttt{Air2} \sqcap \forall \texttt{involved-agent.Program1}$$

2.2 Case Frames

Constraints on complements and modifiers of German words are expressed in terms of case frames which state the valencies of a word and their possible semantic filler types.

$$\begin{aligned} \exists \texttt{involved-agent.TOP} &\sqsubseteq \texttt{Run1} \\ \texttt{TOP} &\sqsubseteq \forall \texttt{involved-agent.Programme1} \end{aligned}$$

The definition encodes the meaning of the German verb *kommen* in the sense of *A film is on in channel TV5.* A GCI axiom is used to define the domain and range of the (thematic) role `involved-agent`. In general, thematic roles are used in a number of case frames, not just in one. This means, more than one GCI has to be included in the linguistic terminology that is used to encode the use of German words that take complments or modifiers. A number of such thematic roles is contained in EuroWordNet terminology. The whole set of roles is listed in table 1. Thematic roles are defined to be features as the relation between discourse referents is functional.

2.3 Mapping DRSs onto ABoxes

The main question to be discussed is the issue of how to verify the mapping of domain independent – in terms of EuroWordNet to application specific language usage – in terms of a domain model. The general idea is that discourse referents in the domain of discourse refer to instances in the application domain. Such pairs of a discourse referent and a corresponding instance are represented by means of a special role called `has-lex`. In the definition $\texttt{AvEvent} \sqsubseteq \forall \texttt{has-lex.Program1}$, it is claimed that an `AvEvent` is related to a discourse referent of `Program1`. As a consequence, all words to whom `Program1` is assigned as meaning in EWN, designate an instance of `AvEvent`. The DRS

$$\left[\frac{\sigma \;\; d}{\texttt{AvEvent}(\sigma)\;\; \texttt{has-lex}(\sigma, d)\;\; \texttt{Program1}(d)}\right]$$

can be mapped onto an ABox asserting

$$\texttt{Program1}(d) \wedge \texttt{has-lex}(\sigma, d) \wedge \texttt{AvEvent}(\sigma)$$

The set $\mathcal{R}$ of possible application specific readings of an instance of `Program1` is the set of all concepts (in the application domain) subsumed by the concept $\forall \texttt{has-lex.Program1}$. In general:

$$\mathcal{R}(\mathrm{D} := \{C : C \subseteq \forall \texttt{has-lex}.D\}$$

The second issue concerns to verify relations between individuals in the application domain from (thematic) roles in a DRS claimed by the parser. In a domain

independent DRS, discourse referents e_1 and e_K may be related by the thematic roles R_1, ..., R_{K-1}:

$$\left[\frac{e_1\ e_2\ \ldots\ e_K}{E_1(e_1)\ R_1(e_1,e_2)\ E_2(e_2)\ \ldots\ R_{K-1}(e_{K-1},e_K)\ E_K(e_K)}\right]$$

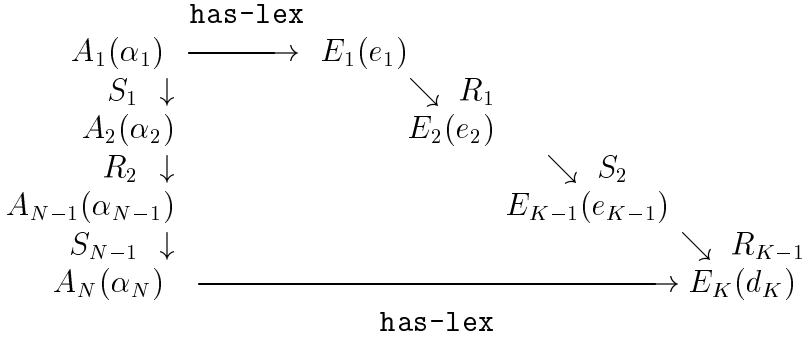
Additionally, we have (from the domain model)

$$\left[\frac{e_1\ \alpha_1\ e_K\ \alpha_N}{E_1(e_1)\ \texttt{has-lex}(e_1,\alpha_1)\ A_1(\sigma_1)\ \texttt{has-lex}(e_K,\sigma_N)\ A_N(\sigma_N)}\right]$$

The case frames for a particular application allow the parser to draw conclusions on roles that hold between α_1, α_2, ..., α_N as a consequence on the assertions made above on e_1, ..., e_K. With description logics, the consistency of these conlcusions may be validated with the help of the following GCI (schema) which is instantiated for each linugistisc valency in the EuroWordNet terminology:

$$\begin{aligned}\exists\texttt{has-lex}.(E_1 \sqcap \exists R_1.(E_2 \sqcap \exists \cdots R_i \cdots.\exists R_{K-1}.(E_K \exists\texttt{has-lex}^{-1}.\texttt{TOP})) \ \rightarrow \\ A_1 \sqcap \exists S_1.(A_2 \sqcap \exists S_2.(A_3 \sqcap \exists \cdots S_j \cdots.\exists S_{N-1} N.A_N))\end{aligned}$$

In this axiom, the R_i are thematic roles, the S_j roles in the application domain. E_i are EuroWordNet concepts, A_j application concepts. The axiom is visualized by the following schema:



In order to explain the "genesis" of the axiom schema above, we have to consider several cases of how thematic roles are mapped onto application specific ones.

2.4 Mapping Thematic Roles onto Pragmatic Ones

The interpretation of thematic roles in terms of roles in the application domain is encoded as the application specific part of the case frames describing the language usage in the application domain as in the following example:

$$\begin{aligned}&\exists\texttt{has-lex}.(\texttt{Program1} \sqcap \\ \exists\texttt{involved-time}.(\texttt{um-tp} \sqcap \exists\texttt{value}.(\texttt{clocktime} \sqcap \exists\texttt{has-lex}^{-1}.\texttt{TOP}))) \ &\rightarrow \\ &\exists\texttt{has-starttime}.\texttt{TimeInterval}\end{aligned}$$

Given two DRSs, with the help of an ABox consistency test, one has to validate the application specific reading constructed by the parser. For the utterance "*Sendung um 20.15*", the parser maps the semantic reprsentation on the left onto its application specific interpretation on the right:

$$\left[\begin{array}{l} d\ t\ c \\ \hline \texttt{Program1}(d)\ \texttt{involved-time}(d,t) \\ \texttt{um-tp}(t)\ \texttt{value}(t,c)\ \texttt{clocktime}(c) \\ \texttt{has-hour}(c,20)\ \texttt{has-minute}(c,15) \end{array}\right] \quad \left[\begin{array}{l} \alpha\ \iota \\ \hline \texttt{AvEvent}(\alpha)\ \texttt{has-lex}(\alpha,d) \\ \texttt{TimeInterval}(\iota)\ \texttt{has-lex}(\iota,c) \\ \texttt{has-starttime}(\alpha,\iota) \\ \texttt{has-hour}(\iota,20)\ \texttt{has-minute}(\iota,15) \end{array}\right]$$

2.5 Instantiating Parameters and is-part-of Relations

Given a thematic role between two discourse referents from two different DRSs where the first one serves as functor and the second one as complifier[1], and the individuals corresponding to the discourse referents, the relation to be validated does not necessarily hold between those two individuals. However, the pragmatic relation may hold between the individual corresponding to the discourse referent selected from the functor and an individual accessible via a role path from the individual corresponding to the discourse referent selected from the complifier's DRS as in the following example (DRS: left, A-Box: right):

$$\left[\begin{array}{l} s\ d \\ \hline \texttt{Recording1}(s)\ \texttt{TVStation1}(d) \\ \texttt{involved-patient}(s,d) \\ \texttt{value}(d,\text{ZDF})\ \texttt{Name}(\text{ZDF}) \end{array}\right] \quad \left[\begin{array}{l} s\ d\ \alpha\ \delta\ \sigma \\ \hline \texttt{Record}(\sigma)\ \texttt{has-lex}(s,\sigma)\ \texttt{has-lex}(d,\delta) \\ \texttt{AvEventLocation}(\delta)\ \texttt{AvEvent}(\alpha) \\ \texttt{has-aveventlocation}(\alpha,\delta)\ \texttt{String}(\text{ZDF}) \\ \texttt{has-value}(\delta,\text{ZDF})\ \texttt{has-avevent}(\sigma,\alpha) \end{array}\right]$$

In this example, the thematic role `involved-patient` is mapped onto the role path $\texttt{has-avevent} \circ \texttt{has-aveventlocation}$ establishing the following the axiom:

$$\begin{aligned} \exists \texttt{has-lex}.(\texttt{Recording1} \sqcap \exists \texttt{involved-patient}.(\texttt{TVStation1} \sqcap \exists \texttt{has-lex}^{-1}.\texttt{TOP})) \quad \rightarrow \\ \texttt{Record} \sqcap \exists \texttt{has-avevent}.(\texttt{AvEvent} \sqcap \exists \texttt{has-aveventlocation}. \\ (\texttt{AvEventLocation} \sqcap \exists \texttt{has-value}.\texttt{String})) \end{aligned}$$

With this GCI, we verify that α is a valid parameter for the action σ and that δ forms a part of the instance of `AvEvent` designated by α.

2.6 Instantiating has-part Relations

Often, it happens that a thematic role between two discourse referents induces more than one relation between individuals in the application domain. For example, in the DRS

[1] *Complifier* means *complement or modifier* as explained in [BLG02].

$$\left[\begin{array}{l} v_1\ v_2\ v3\ v_4\ v_5\ c_3\ c_4 \\ \hline \texttt{TimeDate}(v_1)\ \texttt{has-month}(v_1, c_3)\ \texttt{Month2CalendarMonth1}(c_3)\ \texttt{month-value}(c_3, c_4) \\ \texttt{Number1}(c_4)\ \texttt{value}(c_4, 1)\ \texttt{on}(1) \\ \texttt{has-date}(v_1, v_2)\ \texttt{lc-date}(v_3)\ \texttt{has-day}(v_3, v_4)\ \texttt{Day4}(v_4)\ \texttt{day-value}(v_4, v_5) \\ \texttt{Number1}(v_5)\ \texttt{value}(v_5, 5)\ \texttt{on}(5) \end{array}\right]$$

ABox Assertion	**Large KB**	**Small KB**
(most-specific-instantiators v_4 edge2)	2173	134

Table 2: Performance Evaluation for Racer Version 1.6.3 (in milli seconds)

$$\left[\frac{s\ d}{\texttt{Switch1}(s)\ \texttt{involved-patient}(s, d)\ \texttt{Device1}(d)}\right],$$

two roles have to be instantiated in order to construct the application specific reading: `has-eib` and `has-state`. This may be validated with an GCI axiom:

$$\begin{aligned} \exists\texttt{has-lex}.(\texttt{Switch1} \sqcap \exists\texttt{involved-patient}.(\texttt{Device1} \sqcap \exists\texttt{has-lex}^{-1}.\texttt{TOP})) &\rightarrow \\ \texttt{EIBSwitch} \sqcap \exists\texttt{has-eib}.\texttt{EIBDevice} & \\ \exists\texttt{has-lex}.(\texttt{Switch1} \sqcap \exists\texttt{involved-patient}.(\texttt{Device1} \sqcap \exists\texttt{has-lex}^{-1}.\texttt{TOP})) &\rightarrow \\ \exists\texttt{has-state}.\texttt{ONState}) & \end{aligned}$$

The cooresponding ABox may be represented as follows:

$$\left[\begin{array}{l} s\ d\ \sigma\ \delta \\ \hline \texttt{EIBSwitch}(\sigma)\ \texttt{has-lex}(\sigma, s)\ \texttt{has-eib}(\sigma, \delta) \\ \texttt{EIBDevice}(\delta)\ \texttt{has-lex}(\delta, d) \\ \texttt{has-state}(\sigma, \mathrm{o})\ \texttt{OnState}(\mathrm{o}) \end{array}\right]$$

In such a case `has-state` and `has-eib` are called *has-part* relations, as the thematic role `involved-patient` in the semantic representation of the input utterance leads to the instantiation of the individuals o and δ which are part of the definition for σ.

3 Experiments with RACER

We evaluated the performance of the outlined approach in terms of computation time needed on a Pentium III 800 MHz PC running under SuSE Linux 7.2 with 256 MB of RAM. First, we compiled a big knowledge base consisting of 1165 concept definitions and a large number of additional `disjoint` statements.

This knowledge base contains the complete SUMO ontology encoded in DL, the EWN upper ontology and the concept definitions specific to the EMBASSI applications. The performance test was carried out by parsing the prepositional phrase *am ersten Januar*. At a certain step during the parsing the consistency of the following ABox has to be tested and the most specific concepts of the DRS's head have to be computed. We get the result in table 2.

Of course, more than 2 seconds for a single call to `most-specific-instantiators` is not acceptable for parsing natural language, as – given a quite complex word lattice – hundreds of such calls have to be performed for parsing one lattice under the constraint of real time behavior of the overall system.

Obviously, however, many SUMO and EWN concepts could be deleted from the knowledge base as they were not used by the application specific part of the knowledge base. In an auotmatic precompilation step, we deleted 862 concepts which are only defined, but not used as part of another definition – many among them about insects and bacteria which are not considered relevant for the application. The performance test was then repeated (see table 2).

As the table indicates, RACER performed faster with the smaller knowledge base, and performance becomes practical. However, the ratio of deleted concepts would be worse in more complex domains. As we did not include the whole set of EWN concepts (about 100.000 concepts), in a more complex application the EWN portion even of the reduced knowledge base would be larger. So, faster classification is what we are dreaming of ...

4 Modular Composition of T-Boxes

Constructing and maintaining large knowledge bases is a task for which methods borrowed from software engineering are of help. The most important principles are modularization and decomposition. As mentioned already, in addition to the separation between discourse and application, we maintain two upper models for lexical semantics and the design of application specific domain models. The first one is covered by EuroWordNet, the second one by the Standard Upper Ontology. As EuroWordNet only contains concepts for nouns, verbs, adjectives, and adverbs, we have to complement this linguistic domain model by separate models for defining notions for temporal and spatial expressions – to mention only the most prominent ones.

In this process, software tools such as OilEd and Protegé can support to a very large extent the engineering task of building such libraries of domain models. Their visual presentation of concept hierarchies helps avoiding duplicate definitions as well as forgetting crucial *disjoint* statements. The possibility to detect incoherent definitions by combining editors and DL reasoners improves the quality of the resulting domain models and speeds up their development.

5 Further Requirements for A-Box Reasoning

In general, domain-instantiated A-Boxes are partial models for application operations. Its contents can be modified by user or system messages. As a framework for processing partial information, we found out that FIL [Abd95] 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. Such a separate model could be avoided if hypothetical reasoning could be done with A-Boxes in a direct way: Starting with a "core" A-Box containing "hard", but partial information, alternative extensions with hypotheses have to be computed and checked for consistency and completion. These hypotheses are either already given or have to be asked from the user, which amounts to a mixed-initiative subdialogue. Seen as a tableau, leaves of open branches represent possible information states of the dialogue manager at a given time. Inferences on tableau consistency are drawn using domain concept definitions the constraints of which determine justifications, i.e. possibly underspecified parameters corresponding to optional phrases. In other words, a context mechanism is required which manages alternative contexts consisting of a common core with different extensions depending on particular sets of assumptions. From a technical point of view we have to deal with a belief revision problem which could be handled by a reason maintenance mechanism for A-Boxes (cf. [Neb90, Rei87]).

References

[Abd95] N. Abdallah, *The Logic of Partial Information*. New York: Springer, 1995

[BLG02] K. Bücher, G. Görz, and B. Ludwig, *Corega Tabs: Incremental Semantic Composition*, in this volume

[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

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

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

[Neb90] B. Nebel, *Reasoning and Revision in Hybrid Representation Systems*. Berlin: Springer (LNAI 422), 1990

[NP01] I. Niles and A. Pease. Toward a standard upper ontology. In *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, 2001.

[Vos98] P. Vossen, editor. *EuroWordNet: A Multilingual Database with Lexical Semantic Networks*. Kluwer Academic Publishers, Dordrecht, 1998.

[Rei87] R. Reiter and J. de Kleer, *Foundations of Assumption-Based Truth Maintenance Systems*. Proceedings of AAAI-87, Palo Alto: AAAI, 1987, 183–188

The Use of DL in Component Libraries - First Experiences

Ragnhild Van Der Straeten and Miro Casanova
System and Software Engineering Lab
Vrije Universiteit Brussel, Belgium
Email: {rvdstrae|mcasanov}@vub.ac.be

Abstract

Component-based software development (CBSD) is a very promising software engineering technique for improving reuse and maintenance of software components. However, in order to be able to reuse the components, developers should be aware of the available components in the company, in which contexts those components can be used and how they will behave. Our approach is to use the technique of software libraries to classify the components. The structure of these libraries is specified by ontologies. Those ontologies are specified in the $\mathcal{SHIQ}$ description logic. The reasoning services of DL let us verify whether the ontologies are consistent. This helps automating the process of classifying and retrieving a component into a software library.

1 Problem

Component-based software development (CBSD) is one of the major efforts for improving reusability and maintainability of software applications. A component was defined at ECOOP 96 [SP97] as follows: *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.* The great advantage of component-based software development (CBSD) is that new software can be built by combining bought and self-made components. To do this, it is necessary to specify in which contexts the components can be used and how they will behave.

An important effort for improving reusability of software is the research done on software libraries. A software library is defined as a "managed collection of software assets where assets can be stored, retrieved and browsed" [AM]. One of the main ways of organizing a software library is by using the faceted

classification scheme. In this scheme a facet is a "clearly defined, mutually exclusive, and collectively exhaustive aspects, properties or characteristics of a class or specific subject" [T92]. Each facet is defined by a possible hierarchy of terms. Those terms describe the concrete values the facet can have.

We want to classify components based on the description of the semantics of those components. To establish this classification, we use:

- software libraries using the faceted approach.
- ontologies describing the behaviour of the components. These ontologies will represent a multi-dimensional classification of components and will capture the behaviour of components.

We envision this research as a way of improving reuse because of the use of a library which consists of an organized collection of components together with the relevant documentation. We also envision the improvement of evolution management of component-based applications by having the description of those applications in the library, including the dependencies and other relationships between their constituent components. As ontology language the DL $\mathcal{SHIQ}$ [HS00] has been chosen. In the next section we further explain our research and motivate the choice of the logical formalism. In section 3 examples of these ontologies are given. Section 4 concludes this paper.

2 The Framework

It is our intention to classify components in software libraries by using the faceted approach [PD91] [PF87], allowing a multi-dimensional classification of components. We define a dimension as *a set of facets that are related to the same view or the same aspect of a component.* Different dimensions should be considered in the classification. Examples are the functionality (what it does, inputs, outputs, etc.) of a component, the implementation issues (programming language, platform, etc.) and so on. We use facets and terms as defined in [PD91]. To be able to have a multi-dimensional classification of components we describe the different dimensions, their facets and terms. Furthermore, there are also relevant relations that should be described.

We consider two kinds of relations: inter- and intradimensional ones. The first one are relations between dimensions, the latter one are relations within one and the same dimension. These relations put restrictions on the different dimensions, their facets and the respective terms. Remark that this is one way of classifying the kinds of relations. We are working on a more detailed classification taking into account other aspects. To define these structures we will use different ontologies.

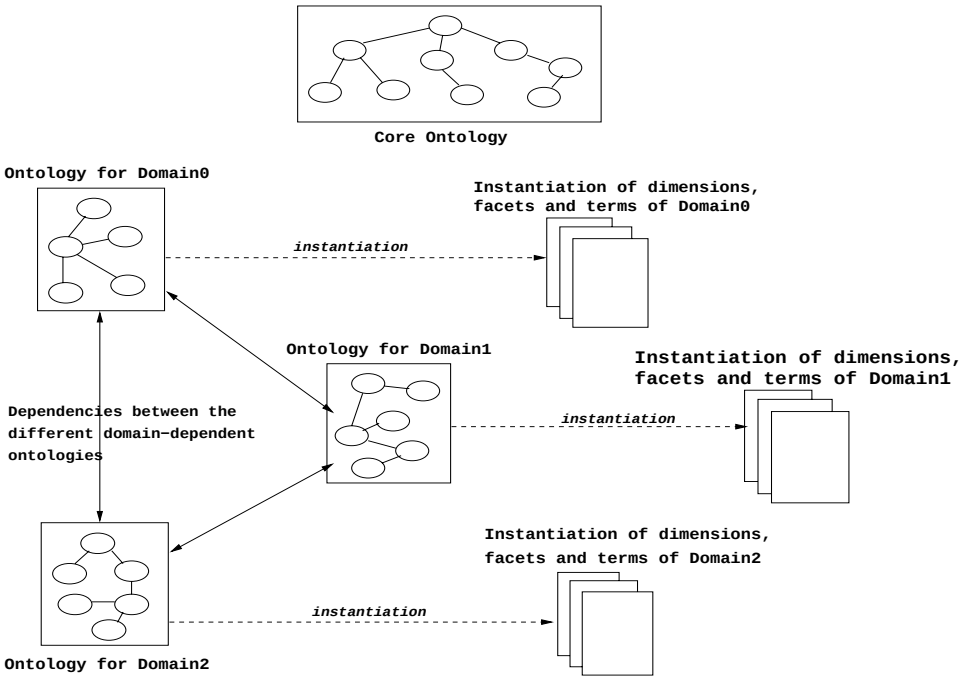


Figure 1: Ontologies for Component Classification

As depicted in figure 1 our framework consists of:

Core Ontology

This ontology consists of dimensions that are relevant for all components. Examples of such dimensions are explained in section 3.1.

Domain-Dependent Ontologies

These ontologies describe the different dimensions relevant for the classification of a given kind of components. Such a kind consist of components belonging to the same domain. Examples of such sets of components are communication network components, user-interface components, data compression components, etc.. Such an ontology will keep track of the facets, terms, dimensions, inter- and intra-dimensional relations of components of one single domain. The advantages of having such ontologies are:

- The knowledge about the components is made explicit by these ontologies.
- The insertion of the components in the library becomes easier because of the explicit presence of the relations. Given a term of a facet, other terms can be automatically derived due to those relations.
- Different libraries based on the same ontology can be easily merged.
- Using ontologies, queries can be executed on software libraries.

As ontology language the very expressive Description Logic $\mathcal{SHIQ}$ [HS00] will be used. The advantages of using Description Logic to build these ontologies are well-known: concepts can be easily composed to form new concepts. DL allows for arbitrary binary relations which enables the expression of the different relations between the components, their terms, facets and dimensions. DL offers efficient reasoning support. This support can be used to reason about and to query the constructed ontology. The specification of these ontologies is terminological expressions.

We remark that next to the inter- and intra-dimensional restrictions within one ontology, we can also have inter-dimensional restrictions between dimensions in different ontologies as indicated in figure 1. We will not consider those relations here because they are still under investigation.

Component Items

They specify the values of the different terms for a specific component. For each component we want to classify, the terms of relevant facets of the different dimensions as described in the ontology are instantiated. Also on this level, we have inter- and intra-dimensional relations. These relations only restrict values concerning these specific component items. Also the instances of the different components and their restrictions will be expressed in DL as assertional knowledge.

In the following section we will give examples of domain-dependent ontologies, component instances and their representation and the reasoning tasks we want to define. The specification of these ontologies and the different dimensions has been figured out by inspecting and analyzing component-based applications and their components. We have to mention that this specification is still evolving.

3 Examples, Representation and Reasoning

3.1 Examples of Dimensions of the Core Ontology

These are dimensions that are relevant for all kinds of components. Two example dimensions are the *implementation* dimension and the *quality of service* dimension. The *implementation* dimension contains information about the implementation of the component such as the platform, the programming language and the compiler (if available). The *quality of service* dimension consists of timing and memory information, or in general it consists of any non-functional requirements or provisions of a component. Its different facets are (so far):

1. Timing provided: a given time-period in which a component will offer a certain service.

2. Timing required: a given time-period within which a component requires some service from another component.

3. Memory required: a given amount of memory a component requires for providing a certain service.

The $\mathcal{SHIQ}$ expressions are shown in figure 2[1].

3.2 Examples of Domain-Dependent Ontologies

An example of such an ontology is the ontology describing the dimensions, facets and terms of network components. In general, the functionality dimension of a component consists of facets that have to do with the functionality part of the component belonging to this domain. In the case of network components, the semantical part of the functionality dimension consists of the following facets and terms:

1. *Kind*: Information about the kind or type of the component.
   - client
   - server
2. *Actions*: the different functions the component can perform. Some terms of this facet are (remark that these lists are not exhaustive):
   - Link
     - Out : reference, subscribe, unsubscribe, connect, disconnect
     - In : referenced, subscribed, unsubscribed, connected, disconnected
   - Data : send, receive, set, get, notify, stream, answer

   An action can have input and output arguments:
   - Input: arguments needed to perform some action.
     - String, Number: {integer, float, ...}, ...
   - Output: results of the performed action.
     - String, Number: {integer, float, ...}, ...

The figure 2 shows a part of this specification in $\mathcal{SHIQ}$.

[1] $\forall^{=1}R.C$ is an abbreviation for $\forall R.C \sqcap (\leq 1R.C) \sqcap (\geq 1R.C)$

3.3 Example of Component Items

Consider a network client component. This component sends strings over a TCP/IP connection and throws events when a string is received or when the connection is established or destroyed. So this component has 4 actions: send, receive, connect and disconnect. The connect action has two input arguments that are both strings (i.e. the port number and the host name), the output of this action is the *rcvConnect* event if the connection succeeds and the *rcvClose* event if the connection fails. This knowledge is specified in Abox assertions.

3.4 Relations and Restrictions

A programming language is related to at least one platform in the implementation dimension. This is an example of an intra-dimensional relation. Another example of such a relation between facets of the implementation dimension, but on the component level is the fact that if the component is compiled using the GNU gcc compiler under Linux, the component's platform must be Linux too. To express this last restriction in DL, we need to be able to express concept inclusion axioms in the Abox as shown in 2. This is possible as explained in [HS01] where $\mathcal{SHIQ}$ is extended with nominals. Consider the network components' ontology again, there exists a relation between the *action* facet of the *functionality* dimension and the *timing provided* facet of the *quality of service* dimension. The *receive* action may take at most 8 seconds. This relation is also written down in figure 2.

3.5 Reasoning

The developer needs the following reasoning services at the ontology level:

1. consistency checking of a domain-dependent ontology,
2. consistency checking of a domain-dependent ontology combined with the core ontology,
3. checking satisfiability of concepts and relations,
4. checking subsumption between concepts or relations.

These reasoning services correspond to the different reasoning tasks of $\mathcal{SHIQ}$. At the component items level, the developer needs support to check the consistency between Abox and Tbox statements and also to check consistency of the Abox. The following remarks can be made concerning the expressiveness needed:

COMPONENT	$\sqsubseteq$	$\forall^{=1}$has_func.FUNCTIONALITY $\sqcap \forall^{=1}$ has_impl.IMPLEMENTATION $\sqcap \forall^{=1}$ has_qos.QUALITYOFSERVICE
IMPLEMENTATION	$\sqsubseteq$	$\forall$has_plat.PLATFORM $\sqcap \forall$has_lang.LANGUAGE $\sqcap \forall$ has_comp.COMPILER
QUALITYOFSERVICE	$\sqsubseteq$	$\exists$timeprovided.$\top \sqcap \exists$timerequired.$\top$ $\sqcap \exists$ memoryrequired.$\top$
FUNCTIONALITY	$\sqsubseteq$	$\forall^{=1}$has_act.ACTION $\sqcap \forall^{=1}$has_kind.KIND
CLIENT	$\sqsubseteq$	KIND
SERVER	$\sqsubseteq$	KIND
ACTION	$\sqsubseteq$	$\forall$has_output.OUTPUT $\sqcap \forall$has_input.INPUT
DATA	$\sqsubseteq$	ACTION
SEND	$\sqsubseteq$	DATA
RECEIVE	$\sqsubseteq$	DATA
IMPLEMENTATION $\sqcap$(= 1 has_lang.LANGUAGE)	$\sqsubseteq$	($\geq$ 1 has_plat.PLATFORM)
COMPONENT $\sqcap \exists$has_func.(FUNCTIONALITY $\sqcap \exists$has_act.RECEIVE)	$\sqsubseteq$	$\exists$has_qos.(QUALITYOFSERVICE $\sqcap \exists(\leq_8$ timerequired))
gcc-Gnu-Linux	:	COMPILER
Linux	:	PLATFORM
$\exists$has_comp.gcc-Gnu-Linux	$\sqsubseteq$	$\exists$has_plat.Linux $\sqcap$ ($\leq$ 1 has_plat)

Figure 2: Definitions of Ontology Concepts and Relations.

- First of all, we need concrete domains to express certain relations. For example, the input arguments of an action can be integers or strings. Unary concrete domains will not be sufficient in this case. In [CL02], $\mathcal{Q}-\mathcal{SHIQ}$ is introduced, which extends $\mathcal{SHIQ}$ with concrete domain concept constructors allowing the representation of rational numbers. However in our case, we need to consider Integer, String and real numbers and it should also be possible to express intervals of real numbers. For example the time required by a certain offered service can be between two real values. The question is of course if decidability is lost if $\mathcal{Q}-\mathcal{SHIQ}$ is further extended.

- It should also be possible to use instances of Tbox concepts, i.e. Abox individuals in Tbox expressions. The name of a certain component can appear in the domain-dependent ontology to which this component can be classified. So we need nominals in our language. In [HS01], $\mathcal{SHIQ}$ is extended with nominals.

4 Conclusion

In this paper we presented our ideas to classify software components in libraries based on ontologies described in the very expressive DL $\mathcal{SHIQ}$. We use Description Logics to specify different ontologies used to classify components. The reasoning services of DL let us verify whether the ontologies are consistent, whether dimensions, facets and terms are satisfiable, whether certain relations or certain concepts in the ontology (i.e. dimensions, facets and terms) subsume each other. This helps automating the process of classifying and retrieving a component into a software library. The next step in this research is to make a more detailed classification of the different possible relations appearing in the ontologies. This will allow us to find out if existing DLs ($\mathcal{SHIQ}$ in particular) are expressive enough for our purposes. Another step is to use a $\mathcal{SHIQ}$ reasoner allowing us to reason about the different ontologies and component items. Finally, we will define queries about the component items in order to retrieve components or specific facets or terms.

References

[SP97] Szyperski, C., & Pfister, C. (1997). Workshop on Component-Oriented Programming, Summary. In MuhlHauser M. (Ed.) Special Issues in Object-Oriented Programming - ECOOP96 Workshop Reader. Dpunkt Verlag, Heidelberg.

[BJ99] Beugnard, J. Jezequel, N. Plouzeau, D. Watkins. "Making Components Contracts Aware". IEEE Computer, July 1999.

[PD91] R. Prieto-Diaz. "Implementing Faceted classification for software reuse". Communications of the Acm. May 1991.

[PF87] R. Prieto-Diaz and Peter Freeman. "Classifying software for reuse". IEEE Software, 4(1):6-16, January 1987.

[AM] S. Atkinson and A. Mili. "Software Libraries", *http://citeseer.nj.nec.com/17413.html*.

[T92] Taylor A. "Introduction to Cataloging and Classification". 8th ed. Englewood, Colorado: Libraries Unlimited, 1992.

[HS01] I. Horrocks and U. Sattler. Ontology Reasoning in the SHOQ(D) Description Logic. In the Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, 2001. Seattle, USA, 2001.

[HS00] I. Horrocks, U. Sattler and S. Tobies. Reasoning with Individuals for the Description Logic $\mathcal{SHIQ}$. In the Proceedings of CADE-00, pg. 482-496, 2000.

[CL02] C. Lutz. Adding numbers to the $\mathcal{SHIQ}$ Description Logic - First Results. In the Proceedings of the Eight International Conference on Principles of Knowledge Representation and Reasoning (KR2002). Morgan Kaufman. Toulouse, France, 2002.