Knowledge representation in process engineering

Franz Baader and **Ulrike Sattler**
RWTH Aachen, {baader,uli}@cantor.informatik.rwth-aachen.de

Abstract

In process engineering, as in many other application domains, the domain specific knowledge is far too complex to be described entirely using description logics. Hence this knowledge is often stored using an object-oriented system, which, because of its high expressiveness, provides only weak inference services. In particular, the process engineers at RWTH Aachen have developed a frame-like language for describing process models. In this paper, we investigate how the powerful inference services provided by a DL system can support the users of this frame-based system. In addition, we consider extensions of description languages that are necessary to represent the relevant process engineering knowledge.

The application domain

Process engineering is concerned with the design and operation of chemical processes that take place in large chemical plants. This engineering task includes activities like deciding on an appropriate flowsheet structure (e.g. configuration of reaction and separation systems), mathematical modeling and simulation of the process behavior (e.g. stating mathematical equations and performing numerical simulations), sizing of components (like reactors, heat exchangers etc.) as well as budgeting and engineering economics.

These highly complex tasks can be supported by building computer models of the chemical plants and processes, using appropriate software tools such CAD, decision support systems and numerical tools. Rather than designing each new model from scratch, one wants a system that offers standard building blocks that can easily be put together. Standard building blocks [Marquardt, 1994; Bogusch & Marquardt, 1995] are objects representing

- material entities such as reactors, pipes, control and cooling units,
- models of these devices such as device-, environment-, and connection-models,
- interfaces between these models and so-called implementations describing their behaviour,
- symbolic equations specifying these implementations and variables occuring in these equations, which are related to each other as specified in the interfaces.

Since there is a great variety of different building blocks, they must be stored in an appropriate database. Since process engineering is a quickly evolving field, the number of standard building blocks increases constantly. Hence it must be possible to define new building blocks in a comfortable way.

The process engineers at the RWTH Aachen we are cooperating with have developed a frame-like language for describing these standard building blocks, [Bogusch & Marquardt, 1995; Marquardt, 1994]. This language allows to group building blocks into classes, and to order the classes in an *is-a/specialization-of* hierarchy. It should be noted that this hierarchy is explicitly given by the person defining the classes (the knowledge engineer), and not automatically inferred from the definition of the class.

As the complexity of the database increases, navigation in the class hierarchy becomes difficult, and modifying or extending the hierarchy becomes dangerous. More precisely, the knowledge engineer is faced with the following prob-

lems:

1. *Finding an old class:* If the knowledge engineer does not know the exact name or even definition of the class (s)he is looking for, navigation in the class hierarchy is difficult, especially in those parts of the database not often used by the knowledge engineer.

2. *Defining a new class:* There is no support for finding the appropriate place in the hierarchy at which the new class should be inserted. This is left to the intuition of the knowledge engineer.

   Often, (s)he may know that A is a subclass of B, but might be uncertain whether the database already contains a more specific subclass B' of B such that A is also a subclass of B'. Because of this uncertainty it often happens that the hierarchy becomes broader than necessary.

   On the other hand, the definition of the new class might be inconsistent, have unintended consequences, or may not be consistent with the place in the hierarchy at which the new class is inserted.

3. *Distributed modeling:* If the class hierarchy is built by different persons simultaneously, then there is a high probability that several classes (with different names) describe the same type of building block in syntactically different terms. This does not only blow up the size of the database; it is also a source for misunderstanding and errors.

The rôle of the DL system

In order to avoid some of these problems, we intend to provide the database with an interface to a description logic system. In principle, the DL system maintains a TBox that contains concept definitions that are obtained from the class definitions of the frame language. Since the frame system does not have a strict formal semantics, and since it provides means like methods, triggers, etc., which cannot be expressed in description logics, these concept definitions can only be approximations of the class definitions. What we require, however, is that the concept hierarchy computed by the DL system coincides with the class hierarchy of the frame system. This is accomplished by an interaction with the knowledge engineer. Whenever (s)he defines a new class, the corresponding concept is classified in the already existing concept hierarchy. If its place in this taxonomy differs from the place at which the knowledge engineer has put the class, then (s)he is notified, and there are two different ways to overcome the problem:

- either (s)he reconsiders her/his decision. This happens if the knowledge engineer notices that the class definition was incorrect.
- or the concept is modified such that classification puts it at the right place. This happens if the class definition was correct, but the corresponding concept does not reflect the class definition entirely due to the restricted expressive power of the description language.

Navigation can now be supported as follows: The knowledge engineer describes—possibly in an incomplete way—the class (s)he is looking for, and then the description logic interface computes the most specific classes subsumed by this description. A closer look at these classes might then reveal how to specialize the description, and the process of going down in the hierarchy can be continued until an appropriate concept is found. The definition of new classes can be supported since the subsumption test of the DL system can compute all equivalent concepts, and then the knowledge engineer can decide whether the newly defined class is redundant or whether its definition must be modified. In addition, the concept descriptions can be tested for unsatisfiability, and the knowledge engineer can be warned whenever unsatisfiability is detected.

The reason for using a DL system in this context was, on the one hand, that this type of knowledge representation languages is rather similar to frame-like languages. On the other hand, DL systems are equipped with subsumption algorithms necessary for providing the envisioned modeling support, as outlined above. In the last decade, a great variety of different description logics has been investigated [Levesque & Brachman, 1987; Nebel, 1988; Schmidt-Schauss, 1989; Patel-Schneider, 1989; Hollunder *et al.*, 1990; Donini *et al.*, 1991; Baader & Hanschke, 1993; De Giacomo & Lenzerini, 1994; Calvanese *et al.*, 1995]. However, the adequate representation of standard building blocks for models in process engineering requires additional expressive power.

Language extensions

The main concern is here to provide appropriate means for describing the *structure* of chem-

ical plants, of the process models, of equations, etc. For this reason, we have investigated part-whole relations (for the vertical representation of structure), and more expressive number restrictions (which can be used to describe horizontal relationships).

Part-whole relations: Since the plants to be modeled are very complex, one should be able both to decompose and to aggregate devices and connections occurring in the plants. A modeling tool should thus be able to support top-down and bottom-up modeling along a sufficiently large number of decomposition levels, or, even better, along any (finite) number of decomposition levels.

In order to represent composite objects correctly, the inference algorithms of the DL system must take the special properties of part-whole relations into account. As in other applications [Gerstl & Pribbenow, 1993; Franconi, 1994; Artale *et al.*, 1994; Pribbenow, 1995], we were thus confronted with the question

- which types of part-whole relations are needed for the appropriate representation of the complex objects in our application. It turned out that objects are decomposed with respect to the component-composite, segment-entity, and member-collection relation, each of them a specialization of the general part-whole relation. Roughly speaking, parts with respect to the member-collection relation are not coupled with each other and are of the same kind, whereas components are coupled with each other in a rather arbitrary way and may be of quite different kinds; finally, segments are of a similar kind, but coupled with each other. Since the knowledge engineer might want to refer to a part, not knowing on which level of decomposition it can be found and with respect to which specific part-whole relation it is obtained, the general *transitive* part-whole relation must also be available.
- how these relations interact. If, in the intuition of the knowledge engineer, the segment-entity relation is transitive, then it must be represented as a transitive role. But what about a component a of a segment b of a whole c: is a also a component of c? Questions concerning these interactions are not yet completely answered, but without an appropriate solution, composite objects cannot be handled appropriately.
- which additional properties concerning the part-whole relation are relevant in the application. For example, the existence of a certain part can be essential for the proper definition of the whole, in contrast to other parts being optional; a part can be exclusive in the sense that it might be a part of at most one object, without the possibility to be shared by other objects; a part can be functional for an object in that this object does no longer work correctly if this part is broken; and many other important properties are conceivable. The appropriate representation of these properties can be quite useful: it allows, for example, to find out whether all essential parts are specified; if this is not the case, the knowledge engineer can be informed, and the missing parts can be determined.

Since at least the general part-whole relation is transitive, the DL system used in this application must be able to handle some kind of transitive relations. Hence, an interesting question is in which ways transitive relations can be included into description languages and how to design appropriate inference algorithms. In [Sattler, 1996], three different extensions of the description language $\mathcal{ALC}$ by transitivity have been investigated.

Number restrictions: As in many other applications, objects in our application are often characterized by the number of other objects to which they are related via a certain relation. For example, we want to describe devices having at least 7 inputs or devices having exactly 5 outputs. In description logics, this kind of knowledge can be expressed using number restrictions, as in

$$(\texttt{device} \sqcap (\geq 7\ \texttt{input})), \quad \text{or}$$
$$(\texttt{device} \sqcap (= 5\ \texttt{output})).$$

This traditional type of number restrictions has rather weak expressive power: the roles occurring in them are atomic, and one can only use fixed numbers (and not variables ranging over numbers). To overcome this deficit, we have investigated various more expressive number restrictions.

In [Baader & Sattler, 1996a], we have introduced so-called symbolic number restrictions, which allow for variables, and can thus be used

to describe concepts like devices having the *same* number of inputs and outputs, as in

$$(\texttt{device} \sqcap (= \alpha\ \texttt{input}) \sqcap (= \alpha\ \texttt{output})),$$

or devices having less inputs than each of their parts have, as in

$$(\texttt{device} \sqcap (= \alpha\ \texttt{input}) \sqcap (\forall \texttt{part}.(> \alpha\ \texttt{input}))),$$

where α is interpreted as some nonnegative integer. The following example reveals a certain ambiguity:

$$\texttt{device} \sqcap (\forall \texttt{part}.(= \alpha\ \texttt{input}) \sqcap (= \alpha\ \texttt{output})))$$

It describes devices where each part has the same number of inputs and outputs. However, it depends on the reading whether different parts can have different numbers of inputs or not. To overcome this ambiguity, we introduced explicit existential quantification of numerical variables (denoted by $\downarrow \alpha$) to distinguish between (1) a device where for each of its parts the number of its inputs equals the number of its outputs and (2) a device where all parts have the same number of inputs and outputs:

$$\texttt{device} \sqcap (\forall \texttt{part}.(\downarrow \alpha.(= \alpha\ \texttt{input}) \sqcap (= \alpha\ \texttt{output}))) \quad (1)$$

$$\texttt{device} \sqcap (\downarrow \alpha.(\forall \texttt{part}.(= \alpha\ \texttt{input}) \sqcap (= \alpha\ \texttt{output}))) \quad (2)$$

Unfortunately, it turned out that the basic inference problems, such as satisfiability and subsumption, are undecidable if this kind of number restrictions is allowed in an unrestricted way. For a restricted language, we have shown that satisfiability is decidable.

Another interesting extension is to allow for *complex* roles in number restrictions. For example, we are interested in describing devices that have at most 7 parts that are components of their components, as in

$$\texttt{device} \sqcap (\leq 7\ \texttt{has-component} \circ \texttt{has-component}),$$

or we want to describe a device that is controlled by the same control unit as all the devices it is connected to:

$$\texttt{device} \sqcap (= 1\ \texttt{control-by} \sqcup \texttt{connect-to} \circ \texttt{control-by}).$$

In these examples, complex roles are built using the operators composition and union of roles.

Other interesting operators are intersection and inversion of roles. Intersection can be used to express, for example, that a devices has at least 2 bidirectional connections:

$$\texttt{device} \sqcap (\geq 2\ \texttt{input} \sqcap \texttt{output}).$$

Inversion comes in if we need the role `part-of` beside the role `has-part`. In [Baader & Sattler, 1996b], it is shown which types of complex roles lead to undecidable inference problems, and for which types of complex roles subsumption and satisfiability remain decidable.

Outlook

This paper describes work in progress. From our cooperation with process engineers, we have learned that the system services provided by DL systems appear to be very useful for their application: a description logics based browser could support the engineers in building and maintaining their frame-based database. Before building this browser, we have investigated which expressive power is needed to describe relevant properties of objects occuring in this application. It turned out that transitivity and expressive number restrictions play an important rôle in this application. However, we have not yet made a final decision as to which particular description logic is "most appropriate". We need to find a compromise between necessary expressive power and acceptable computational complexity. In the near future, such a description language will be fixed and we will test whether the support DL systems can provide in this application is really as high as we expect.

References

[Artale *et al.*, 1994] A. Artale, F. Cesarini, E. Grazzini, F. Pippolini, and G. Soda. Modelling composition in a terminological language environment. In *Workshop Notes of the ECAI Workshop on Parts and Wholes: Conceptual Part-Whole Relations and Formal Mereology*, pages 93–101, Amsterdam, 1994.

[Baader & Hanschke, 1993]
F. Baader and P. Hanschke. Extensions of concept languages for a mechanical engineering application. In *Proc. of the 16th German AI-Conference, GWAI-92*, volume 671 of *LNCS*, pages 132–143, Bonn, Deutschland, 1993. Springer-Verlag.

[Baader & Sattler, 1996a] F. Baader and U. Sattler. Description logics with symbolic number restrictions. In W. Wahlster, editor, *Proc. of ECAI-96*. John Wiley & Sons Ltd, 1996.

[Baader & Sattler, 1996b] F. Baader and U. Sattler. Number restrictions on complex roles in description logics. In *Proc. of KR-96*. M. Kaufmann, Los Altos, 1996.

[Bogusch & Marquardt, 1995] R. Bogusch and W. Marquardt. A formal representation of process model equations. *Computers and Chemical Engineering*, 19:211–216, 1995.

[Calvanese *et al.*, 1995] D. Calvanese, G. De Giacomo, and M. Lenzerini. Structured objects: Modeling and reasoning. In *Proc. of DOOD-95*, volume 1013 of *LNCS*, pages 229–246, 1995.

[De Giacomo & Lenzerini, 1994] G. De Giacomo and M. Lenzerini. Concept language with number restrictions and fixpoints, and its relationship with mu-calculus. In *Proc. of ECAI-94*, 1994.

[Donini *et al.*, 1991] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In *Proc. of KR-91*, Boston (USA), 1991.

[Franconi, 1994] E. Franconi. A treatment of plurals and plural quantifications based on a theory of collections. *Minds and Machines*, 3(4):453–474, November 1994.

[Gerstl & Pribbenow, 1993] P. Gerstl and S. Pribbenow. Midwinters, end games and bodyparts. In N. Guarino and R. Poli, editors, *International Workshop on Formal Ontology-93*, pages 251–260, 1993.

[Hollunder *et al.*, 1990] B. Hollunder, W. Nutt, and M. Schmidt-Schauss. Subsumption algorithms for concept description languages. In *ECAI-90*, Pitman Publishing, London, 1990.

[Levesque & Brachman, 1987] H. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.

[Marquardt, 1994] W. Marquardt. Trends in computer-aided process modeling. In *Proc. of ICPSE '94*, pages 1–24, Kyongju, Korea, 1994.

[Nebel, 1988] B. Nebel. Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383, 1988.

[Patel-Schneider, 1989] P. F. Patel-Schneider. Undecidability of subsumption in NIKL. *AIJ*, 39:263–272, 1989.

[Pribbenow, 1995] S. Pribbenow. Modeling physical objects: Reasoning about (different kinds of) parts. In *Time, Space, and Movement Workshop 95*, Bonas, France, 1995.

[Sattler, 1996] U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *20. Deutsche Jahrestagung für Künstliche Intelligenz*, volume 1137 of *LNAI*. Springer-Verlag, 1996.

[Schmidt-Schauss, 1989] M. Schmidt-Schauss. Subsumption in KL-ONE is undecidable. In *Proc. of KR-89*, pages 421–431, Boston (USA), 1989.