A Description Logic for Composite Objects for Domain Modeling in an Agent-Oriented Application

Patrick Lambrix
Department of Computer and Information Science
Linköping University
S-581 83, Linköping, Sweden
patla@ida.liu.se

Lin Padgham
Department of Computer Science
Royal Melbourne Institute of Technology
Melbourne, VIC 3000, Australia
linpa@cs.rmit.edu.au

1 Introduction

In this paper we present a description logic for composite objects and show its use for domain modeling in a particular application. The description logic extends standard description logics by providing support for representation of and reasoning about part-of. We have extended the standard reasoning tasks of subsumption, classification and recognition to include knowledge about part-of. Further, we maintain a part-of hierarchy for individuals that can be used for querying the system about composite objects. We provide the system with new user functions to allow for traversing the part-of hierarchy as well. Our extended description logic system retains the well-known advantages of description logic systems while extending their representational and inferential capabilities.

The application involves automating the monitoring of the Reaction Control System (RCS) of NASA's space shuttle. The application has been modeled before and an implementation exists using the agent-oriented system dMARS (distributed Multi Agent Reasoning System) from the Australian Artificial Intelligence Institute. Although it was recognized that part-of played an important role in the application, the existing implementation did not provide any support in representing and inferencing with part-of. The use of our description logic for composite objects has provided a number of advantages.

In the following section we briefly describe dMARS and the Reaction Control System application (section 2). We propose our description logic for composite objects and describe the advantages that the use of this description logic gave us for the RCS application in section 3. The paper concludes in section 4.

2 dMARS and RCS

dMARS is a situated reasoning system that was built for real-time applications. Each dMARS agent consists of the following components: a database of current beliefs or facts about the world, a set of current goals to be realized, a set of plans describing how a sequence of actions may achieve a given goal or forming a reaction to a particular situation, and an intention structure containing a set of plans that have been chosen for eventual execution. An inference mechanism selects appropriate plans on the basis of the agent's beliefs and goals, and places these selected plans in the intention structure and executes them. The system interacts with its environment through its database by acquiring new beliefs and through the actions it performs.

The database typically contains both static and dynamic information. The static information describes the structural model of the domain, such as what (types of) objects there are, and how they are related. The dynamic information consists of variables which are modified as the world changes or is believed to have changed. An agent typically believes the world has changed (and updates its database), when it takes some action designed to produce change in the world, or when it is notified (by some other agent) that something has happened.

Plans consist of a body, which describes the different steps in the procedure, and an invocation consisting of a triggering part and a context condition part. The triggering part describes the events that must occur for the plan to be executed. They can be the acquisition of a new goal or some change in the database. The context condition describes conditions which must be satisfied in the database. The set of plans in an application not only contains specific knowledge about the application domain, but also meta-level plans containing information about how the beliefs, goals and intentions are to be manipulated. An example of such a meta-level plan is a plan that chooses a plan from a set of plans. The dMARS decides which plans are applicable by matching beliefs and goals with invocation conditions by unification.

The problem domain which we have worked on is the Reaction Control System of NASA's space shuttle. A space shuttle has three RCSs, two aft and one forward. An RCS provides propulsive forces from a collection of jet thrusters to control the attitude of the space shuttle. The RCS modules contain a collection of jets, a fuel tank, an oxidizer tank, two helium tanks, feedlines, manifolds and other supporting equipment. Each RCS module receives all commands via the space shuttle flight software. The various valves in an RCS module are controlled from a panel of switches and talkbacks. The talkbacks pro-

<*concept-descr*> ::=
⊤ | ⊥ | <*concept-name*>
| (**and** <*concept-descr*>$^+$)
| (**one-of** <*individual-name*>$^+$)
| (**all** <*role-name*> <*concept-descr*>)
| (**atleast** <*positive-integer*> <*role-name*>)
| (**atmost** <*non-negative-integer*> <*role-name*>)
| (**fills** <*role-name*> <*individual-name*>$^+$)
| (**allp** <*part-name-name*> <*concept-descr*>)
| (**atleastp** <*positive-integer*> <*part-name-name*>)
| (**atmostp** <*non-negative-integer*>
<*part-name-name*>)
| (**part-fills** <*part-name-name*>
<*individual-name*>$^+$)
| (**module-fills** <*individual-name*>$^+$
<*part-name-change*>*)
| (**pp-constraint** <*role-name*>
<*part-name-name*> <*part-name-name*>)
<*concept-name*> ::= <*symbol*>
<*individual-name*> ::= <*symbol*>
<*role-name*> ::= <*symbol*>
<*part-name-name*> ::= <*symbol*>
<*part-name-change*> ::=
<*part-name-name*> → <*part-name-name*>

Figure 1: Syntax

$\varepsilon[\top] = \mathcal{D}$
$\varepsilon[\bot] = \emptyset$
$\varepsilon[(\textbf{and } A\ B)] = \varepsilon[A] \cap \varepsilon[B]$
$\varepsilon[(\textbf{one-of } i_1 \ldots i_m)] = \{\varepsilon[i_1], \ldots, \varepsilon[i_m]\}$
$\varepsilon[(\textbf{all } r\ A)] = \{\ x \in \mathcal{D} \mid$
$\forall\ y \in \mathcal{D}$: $<x,y> \in \varepsilon[r] \rightarrow y \in \varepsilon[A]\}$
$\varepsilon[(\textbf{atleast } m\ r)] = \{\ x \in \mathcal{D} \mid$
$\sharp\ \{\ y \in \mathcal{D} \mid <x,y> \in \varepsilon[r]\} \geq m\ \}$
$\varepsilon[(\textbf{atmost } m\ r)] = \{\ x \in \mathcal{D} \mid$
$\sharp\ \{\ y \in \mathcal{D} \mid <x,y> \in \varepsilon[r]\} \leq m\ \}$
$\varepsilon[(\textbf{fills } r\ i_1 \ldots i_m)] = \{\ x \in \mathcal{D} \mid$
$<x,\varepsilon[i_1]> \in \varepsilon[r] \wedge \ldots \wedge <x,\varepsilon[i_m]> \in \varepsilon[r]\ \}$
$\varepsilon[(\textbf{allp } n\ A)] = \{\ x \in \mathcal{D} \mid$
$\forall\ y \in \mathcal{D}$: $y \lhd_n x \rightarrow y \in \varepsilon[A]\}$
$\varepsilon[(\textbf{atleastp } m\ n)] = \{\ x \in \mathcal{D} \mid$
$\sharp\ \{\ y \in \mathcal{D} \mid y \lhd_n x\ \} \geq m\ \}$
$\varepsilon[(\textbf{atmostp } m\ n)] = \{\ x \in \mathcal{D} \mid$
$\sharp\ \{\ y \in \mathcal{D} \mid y \lhd_n x\ \} \leq m\ \}$
$\varepsilon[(\textbf{part-fills } n\ i_1 \ldots i_m)] = \{\ x \in \mathcal{D} \mid$
$\varepsilon[i_1] \lhd_n x \wedge \ldots \wedge \varepsilon[i_m] \lhd_n x\ \}$
$\varepsilon[(\textbf{module-fills } i_1 \ldots i_m$
$n_{11} \rightarrow n_{12} \ldots n_{k1} \rightarrow n_{k2})] = \{\ x \in \mathcal{D} \mid$
$\varepsilon[i_1]\ \lhd_{\textbf{mod } \{n_{11} \rightarrow n_{12}, \ldots, n_{k1} \rightarrow n_{k2}\}}\ x \wedge \ldots$
$\wedge\ \varepsilon[i_m]\ \lhd_{\textbf{mod } \{n_{11} \rightarrow n_{12}, \ldots, n_{k1} \rightarrow n_{k2}\}}\ x\ \}$
$\varepsilon[(\textbf{pp-constraint } r\ n_1\ n_2)] = \{\ x \in \mathcal{D} \mid$
$\forall\ y_1, y_2 \in \mathcal{D}$: $(y_1 \lhd_{n_1} x \wedge y_2 \lhd_{n_2} x) \rightarrow$
$<y_1, y_2> \in \varepsilon[r]\}$

Figure 2: Semantics

vide feedback on the position of their associated valves. The aim of the RCS application is to automate the malfunction procedures for the RCS.

Each RCS is managed by two dMARS agents. The Interface agent handles all information concerning transducer readings, valve switches and valve talkbacks. The database for this agent contains the domain model including the knowledge about transducer readings and switch and talkback positions. It has no knowledge about pressures as these have to be deduced from the transducer readings. The plans are typically about calculating pressures, switch movements and malfunction detection. The Controller agent takes a high-level view of the application. The database is similar to the database of the Interface agent, but contains information about pressures rather than about specific transducer readings. This information is obtained by asking the Interface agent. The plans contain the high-level malfunction procedures as they appear in the shuttle's malfunction handling manuals.

For more information about how the RCS application is modeled in the existing dMARS application we refer to [GI89; GI90].

3 Domain Model Using a Description Logic for Composite Objects

As a framework for the modeling of this application we used a description logic with syntax and semantics as described in figures 1 and 2. The non-standard constructs **allp**, **atleastp**, **atmostp** and **part-fills** are part name analogues of well-known standard constructs for roles. The **pp-constraint** construct allows for defining constraints between parts. The **module-fills** construct allows for defining modules. An individual i_1 is a module of another individual i_2 if all the parts of i_1 are also parts of i_2, but i_2 has more parts. The part name changes allow for changing the names of parts from the module to the more composite individual. For a discussion on the underlying part-of model and more discussion and examples with respect to the language we refer to [Lam96]. We also maintain a part-of hierarchy for individuals and have implemented a number of user functions that allow easy traversal of this hierarchy. The system is implemented as an extension to CLASSIC. For details we refer to [Lam96].

In modeling the RCS system we have used both the model as it is in the existing dMARS implementation, and the NASA manual [Bus87] describing the RCS of the space shuttle and its operation, which was the original source document for the application. We have modeled concepts such as jets, tanks, valves, switches and the necessary instances of these concepts. The resulting knowledge base contains 44 concepts and 153 individuals with 38 different part-whole relations and 23 other relations. There are between one and sixteen individuals for each concept. This model closely follows the original source document.

The well-known advantages of using a description logic system for maintaining a knowledge base such as automatic classification, logical inferencing and consistency checking were observed in this application as well. These

advantages were extended with representation of and reasoning about part-of. Further, we found a number of other advantages, specific for part-of, in this application as well. We briefly describe these advantages.

In a natural model of the RCS application the part-of relation plays an important role in the description of the system. The original model contained much information about part-of and the queries in many of the plans of the application relied on the composite nature of objects. In our system there is support for part-of by allowing the distinction between part-of and other relations, and among different kinds of part-whole relations, by allowing domain restrictions and number restrictions, and by allowing for constraints between parts. The previous implementation contained significant information about part-whole relations, but this could not be represented in a standard manner. For instance, the relation between a system and its oxidizer sub-system was represented by 'part-of' and 'oxidizer-subsystem'. The first relation was used in plans when any sub-system could be used in the unification process, while the other relation required the sub-system to be the oxidizer sub-system. In our model we used the part name 'oxidizer-subsystem' which by definition then has the part-of intuition. In a situation where different sub-systems could be used, we can use the description logic system functions to find all possible sub-systems.

In the previous model several new relations existed that did not occur in the original description of the application. These were mainly added for efficiency reasons, in order to skip some unification steps. They usually involve individuals in different sub-systems that have similar functions or individuals where one individual operates the other. An example of the latter is the case of valves, switches and talkbacks that are connected. In the previous model new relations (such as associated-switch) were introduced between these individuals. In our system these extra relations are not needed. We can simply traverse the part-of hierarchy for the composite object to find the relevant individual(s). For instance, in the previous model a valve was always connected to its corresponding switch, by a relation associated-switch. We can find the correct switch without the introduced relation associated-switch, simply by accessing the switch that is part of the same composite individual as the valve.

In the case where additional relations may be desirable for extra efficiency we have used the **pp-constraint** construct to automatically introduce these relations between the different individuals. In the case of valves, switches and talkbacks, for instance, we introduced the concept of valve-switch-talkback-system. The definition of this concept contains the **pp-constraint** that the valves and the talkback must be in the associated-switch relation with the switch. Each collection of connected valves, switches and talkbacks make up one such valve-switch-talkback-system. When a valve-switch-talkback-system is instantiated with the specific individuals, the description logic system makes sure that the associated-switch relations are also maintained.

When adding the various individual components of the system we found that the notion of modules allowed for a convenient "bottom-up" building of objects. We were able to first instantiate the "smallest" composite individuals, such as a connection, and then use the **module-fills** construct to include these in more complex composite individuals, such as an assembly, whereupon the constraints are checked automatically and values are propagated.

The use of modules was shown to be a natural choice as well. Some levels in the part-of hierarchy which existed in the NASA manual did not exist in the previous model of the application. These levels were usually levels representing modules in our part-of hierarchy.

In addition to modeling the world within the description logic system, it was necessary for us to rewrite the system plans, in order to query the description logic knowledge-base regarding the state of the world rather than performing unification on the dMARS representation of the world state. This was relatively straightforward and in some cases resulted in a conceptual simplification of the plans produced. A significant number of the queries needed relied on the representation of individuals as composite entities made up of parts, thus justifying our choice of the extended description logic rather than the simpler unmodified CLASSIC system. The following types of queries appeared in the plans and were all frequently used. Is individual i' part of individual i? Is individual i' part of an individual belonging to C? Is there an individual belonging to C that is part of individual i? Get all individuals that are part of individual i. Get all individuals of which individual i' is a part. Get all individuals that belong to concept C and are part of individual i. Get all individuals that belong to concept C and of which individual i' is a part.

It is also worth noting that we often require only one query, whereas the previous dMARS model typically had to perform the query in several unification steps introducing intermediary variables. One reason for this is the fact that our system allows us to state complex queries regarding part-of.

We found that the checking of the context condition in the dMARS plans can often be split into two conceptually separate phases. In the first phase the actual context condition is checked, i.e. the requirements that have to hold for this plan to be instantiated. The second part of the context conditions then instantiates different variables that are not part of the plan instantiation requirements but for which a plan has to be instantiated for each possible binding.

An example of this is a plan for updating the reading of the quantity in a tank with bad pvt-status by using its associated helium tank. Instead of having nine clauses, binding five different variables, some of which are only intermediate variables for passing between clauses, we were able to simply form two queries - the first ascertaining whether the invocation condition for the plan was met (the given tank is a propellant tank with bad pvt-status and is included in a propellant system for which

the other-propellant-system also has a tank with bad pvt-status) and the second to ascertain which individual helium tank is included in the same propellant system as the original tank. It was also possible to define the body of this plan without the use of additional variables.

In the process of building the knowledge base, some mistakes were found in the original database on which we were working.[1] A number of these errors would have been automatically detected, or more easily noticed, using a description logic for composite objects. Examples of types of errors detected include the following: information appearing twice - a description logic system detects the fact that the information already exists, and does not add the redundant fact; typing mistakes in some relation names, resulting in the relations being undefined - a description logic system issues a warning when it creates the new (mistyped) entity; mismatch between concepts in plans and concepts in the database - a description logic system detects an error; some connection relations were missing - the description logic system can check this by using a system function to check whether all (part-of) relations are closed, i.e. whether all necessary relations are completely instantiated; some of the extra relations were missing - the way that we modeled this, these are automatically created by the description logic system, because they are defined as being necessary for all individuals of the given type.

Although the errors were either corrected in a later version, or were unimportant for the correct functioning of the system, it is clearly beneficial to have support which minimizes such problems.

4 Conclusion

We have presented a description logic for composite objects and described the advantages for a particular application. We saw that a natural model of the application needed part-of. This is obvious when we notice that the model actually contains 38 different part-whole relations. The fact that the model is closer to the original source document than the existing dMARS model comes mainly from the fact that we used part-of. For instance, we did not need to introduce different part-whole relations between the same two individuals. ¿From a logical point of view we did not either need to introduce extra relations that were introduced in the existing dMARS model as traversing the part-of hierarchy would give us the same information. When these extra relations were desirable for efficiency reasons, we used the **pp-constraint** construct to automatically maintain these relations. Using this approach we were able to correct some mistakes found in the existing database. The **module-fills** construct allowed for a bottom-up instantiation of composite individuals, where more complex individuals were instantiated using less complex individuals. We saw that the notion of module was also present in the original source document, but had disappeared in the existing dMARS model. Many of the queries used in the plans involved the part-of relation. Therefore it was natural to use a system where knowledge about part-of can be expressed in a natural way. The user functions of our description logic system also enabled an easy way of traversing the part-of hierarchy.

[1] Here we used the database as in [GI90]. Newer versions of the database exist.

Acknowledgements

The authors would like to thank David Kinny, Ralph Rönnquist and Mike Georgeff from the Australian Artificial Intelligence Institute for useful discussions on the RCS application. This work was done while the first author was visiting the Computer Science Department of the Royal Melbourne Institute of Technology. The first author is supported by grant 95-176 from the Swedish Research Council for Engineering Sciences (TFR).

References

[Bus87] Bush, R., *Reaction Control System Training Manual RCS 2102*, Flight Training Branch, Missions Operations Directorate, NASA, Johnson Space Center, Houston, TX, April 1987.

[GI89] Georgeff, M., Ingrand, F., 'Decision-Making in an Embedded Reasoning System', *Proceedings of IJCAI 89*, pp. 972-978, 1989.

[GI90] Georgeff, M., Ingrand, F., 'Research on Procedural Reasoning Systems - Final Report - Phase 2', SRI International, 1990.

[Lam96] Lambrix, P., *Part-Whole Reasoning in Description Logics*, Ph.D. Thesis 448, Department of Computer and Information Science, Linköping University, 1996.