Reasoning in a Closed Terminology

François Rousselot, François de Beuvron, Michael Schlick
and **David Rudloff**
LIIA, ENSAIS
24, bd de la Victoire, F-67084 Strasbourg Cedex
rousse@liia.u-strasbg.fr

Abstract

The paper presents an extension of Description Logic to cope with configuration problems. In the paradigm of non structural algorithms the definitions are different from classic one. The study originated by Weida [3] in 1992 with CLASSIC and by others [2] (PROSE at AT&T) must be reviewed in order to take account of the extended expressiveness of the language and of its semantic.

The paper describes what is intended by terminological closure, a way to modify and constrain the terminology in order to allow the special kind of reasonning needed in the framework of configuration.

The implementation is presented: it is based on the introduction of special indicators that are manipulated easily, in a uniform way in the tableaux-based inference engine.

Indicators are convenient because they allow to be constructed when concepts definitions are interpreted and the process can be applied on general forms of concepts definitions.

The use of multi TBoxes gives a help for modelling and decreases the computing complexity by partitioning the research space. The application made with the system of the LIIA named CICLOP is applied to the configuration of the electronic security systems in a car.

1 Introduction

Terminological logic can be tuned to provide a good way in facilitating the configuration task. This paper shows how it is realized in CICLOP, the DL system developed by the LIIA. The aim of the laboratory is to use its tool to solve real industrial problems, so the idea to write an extension to cope with configuration problems on a kernel DL system appeared to be a good one. CICLOP DL is based on the tableaux calculus and has a good expressiveness (ALCFIHR+ with axioms) joined to an interesting efficiency. (See DL99 System comparison [5]). The idea to use DL to help configuration comes from the study originated by Weida [3] in 1992 with CLASSIC and by others [2] (PROSE at AT&T).

The present paper is based on Weida's ideas concerning configuration modelling and concerning DL use. It also introduces several improvements due to the necessity to be more precise when replacing structural algorithms with tableaux calculus in order to cope with more expressive Description Logics.

2 Configuration: general principles

A terminology used for configuration is specific, it aims at representing classes of objects to configure i.e. to found subparts which respect some constraints. So the knowledge base must be carefully designed following certain modelling principles. For example, the knowledge base must describe the classes of main objects to configure, the classes of sub-objects which can be parts of main objects and the constraints between the different parts. The important relation is part-of and the important constraints are generally spatial. Presently, in the CICLOP system spatial reasoning [7] is not possible, so it is modelled by using a numerical concrete domain i.e. by using numerical values of attributes (roles which can have only one value) and a linear constraint solver (see 5).

The concepts of the terminology can be divided in two classes: abstract concepts and concrete concepts. Concrete concepts are concepts corresponding to real world entities, abstract concepts are situated on a higher level and represent abstractions for instance: functionality choices or general choices. Since they correspond to real objects, instances of concrete concepts must be completely defined. Unfortunately, if DL are well adapted to manipulate incomplete knowledge, it is not so easy to be sure that an instance is completely defined. So it is necessary to modify the terminology in the way presented in section 3.

Configuration modelling

As concrete concepts are well defined, some hypotheses on their definitions can be done. Modelling the knowledge of configuration problems has to be done very carefully. OR operators must be the less frequent as possible. Concrete concepts inherit from abstract concepts. In the hierarchy, as already seen, OR operators describe possible options and choices at higher levels. Generally OR between primitive concepts are used. Fortunately in real problems, there are not many choices. OR between role expressions are forbidden, OR in role expression are allowed between role domain constraints. Consequences of this restriction is to be seen in section 3.

Configuration in DL

Intuitively, configuration consists of the refinement of an incomplete instance until every of its components is complete. The process starts with one very abstract instance representing global abstract choices for the object or for the system to configure. This initial instance is then completed with other assertions in order to refine the previous choices. The refinement process goes on until there will be no more choices available. To guide the system, a slightly extended DL can verify at every step if the choices are compatible with the constraints of the terminology. Moreover, it can propose to the users the possible choices before he takes his decision and then it propagates the consequences of these choices.

Configuration process is sketched in section 4, it is based on the computing of several concept sets related to a certain instance I which represents the object to be built .

3 Terminological closure revisited

Terminological closure is a transformation on the terminology realized to allow configuration. As soon as knowledge of the domain are considered to be stable. What is wanted are ways of ensuring that an instance is complete and that it is not over-specified.

Vocabulary closure

It is the first thing to do. So in this stage the set of concept names, the set of relations names are closed. New declarations, new concept names introductions are prohibited.

Domain closure

It is the next step. This modification's outcome is that the union of the extensions of all the concrete concepts covers the whole interpretation domain. It means that the extension of every abstract concept corresponds to and only to the union of the extensions of a certain set of concrete concepts.

Syntactic closure of the terminology

It is the last step. Its aim is to prepare concrete concepts to force their instances to inherit exactly and exclusively from their super concepts. It must be encoded that an instance uses all and only all its inherited primitive concepts and all and only all its roles. Compatangelo proposed an analogous transformation with a different goal, the modelling of flow in a plant [4].

Forcing values

The final step is to conceive a way to force the roles of the instance to have values.

Formalisation

Domain closure

Given the set $C_1, C_2, \ldots, C_n$ of concrete concepts,

we have $\Delta = \sqcup_{i=1,n} C_i$

Δ is inserted between $\top$ and its sub concepts, so for every C of the terminology, Δ subsumes C i.e. $C \sqsubseteq \Delta$. Indeed the consistency of the modified terminology has to be proved.

Syntactic closure

Given C_{pT} the set of the primitive concepts of the terminology T and given R_T the set of roles used in T. Suppose for the moment that the expansion of the definition of a given concrete concept C, don't contain ORs.

So C can be set in the form $C = C_{cp} \sqcap R_p$
where C_{cp} is the primitive concept part and R_p the role part.

The primitive concept part is a conjunction of primitive concepts or negated primitive concepts.

The role part is a conjunction of expressions $R_{C,k}$.

Every $R_{C,k}$ is a conjunction of expressions of the form $O_k\ r_k\ c_k$ where O_k is a member of $\{ALL, SOME, \ldots\}$

We defined R_C, the set of roles used by C, as the set of r_k

we have $R_C = \{r_1, \ldots, r_i, \ldots, r_m\}$
where every r_i occurs in at least one expression $O_k\ r_k\ c_k$

To ensure that such a concept cannot be specialized, we have to add to its expression the negation of the complementary of its primitive concept set and the negation of the complementary of its role set.

$C_C = \{P_1, \ldots, P_n\}$
is the set of the concepts used in C, where P_i is a primitive concept used in C.

Notice that in the expression of C either P_i or $(NOT P_i)$ can appear.

As a matter of fact, if P_i or $(NOT P_i)$ appears in the definition of a concept, this is the mark of a choice and P_i must be present. For example, if $(NOT P_i)$ appears and P_i is added, it will clash following the standard clash rule, so only $(NOT P_i)$ is consistant.

The set of the primitive concepts not used in C is CP_C where $CP_C = C_{pT} \setminus C_C$

we call $\overline{C_{p,C}} = \sqcap_{C_i \in CP_C} \neg C_i$ the intersection of every concept of the set of the primitive concepts not in C

The set of the roles not used in C is R_C, where R_C is $\overline{R_C} = R_T \setminus R_C$.

We replace C by $C \sqcap \overline{C_{p,C}} \sqcap_{r_i \in \overline{R_C}} (\forall r_i.\bot)$ in the terminology.

Every expression $(\forall r_i.\bot)$ ensures that r_i cannot be used.

Roles of concrete concepts must have a value (value forcing), we had to force it now. For every role r_i, if it has not a restriction entailing the existence of at least one value, we add $\exists r_i \Delta$.

general case

In the general case, a concept expression is put in a disjunctive form i.e. it contains ORs between conjunctives expressions. In the following , we only consider the OR operators at the first level and in every conjunction the AND at the first level. The generalisation of the processing just described will be its application successively to every disjunctions. In particular, when there is a disjunction: there are as much sets of primitive concepts used in the concrete concept and sets of role use as there are conjunctives expressions.
The implementation part proposes a functional transformation of the terminology that garanties that our modelling hypotheses will be respected.

implementation

Two ways are presently being explored: on the one hand, the use of "indicators" which permit to act as if the lists of primitive concepts and of roles are intensional, on the other hand to use an other TBox in order to maintain meta-information about every concrete concept. Only the use of indicators is shown here.

First of all, the domain had to be closed. This is operated in a simple way. A meta-constraint is introduced as an axiom in the LD. (The term meta-constraint is employed by Horrocks [1]). Every concept C introduced in the terminology is intersected with the following expression: $(OR\ C_1(OR\ C_2(OR\ C_3(OR \ldots (OR\ C_i))) \ldots))$ where all C_i are concrete concepts.

Like in Horrocks this meta-constraint is optimised by using "semantic branching".

Indicators are new operators introduced to solve two problems: firstly to assume the formalisation previously seen, secondly to be used in our tableaux algorithm without modifying it.

The method using indicators implies a transformation of the terminology. First of all, new primitive atomic concepts must be added to the terminology. More precisely, to every primitive concept P of the terminology, a primitive atomic concept $Ind(P)$ is associated. Similarly, to role R of the terminology is associated a primitive concept $Ind(R)$.
The function f that maps now the terminology into the new one is defined in the following manner. if C_1 , C_2 C are some concept expressions, P a primitive concept and r some role:

$$
\begin{array}{ll}
(AND\ C_1\ C_2) & \mapsto (AND\ f(C_1)\ f(C_2)) \\
(OR\ C_1\ C_2) & \mapsto (OR\ f(C_1)\ f(C_2)) \\
P & \mapsto (AND\ P\ Ind(P)) \\
(ALL\ r\ C) & \mapsto (AND\ (ALL\ r\ C)\ Ind(r)), \\
(SOME\ r\ C) & \mapsto (AND\ (SOME\ r\ C)\ Ind(r)) \\
(NOTP) & \mapsto (OR(NOTP)\ (NOTInd(P))) \\
(NOT(ALL\ r\ C)) & \mapsto (OR(SOME\ r\ (NOTC)) \\
 & \qquad (NOTInd(r))) \\
(NOT(SOME\ r\ C)) & \mapsto (OR(ALL\ r\ (NOTC)) \\
 & \qquad (NOTInd(r)))
\end{array}
$$

Finally, to every primitive concept P_i not occurring in C, the primitive concept $(NOTInd(P_i))$ is added.
Given CP_C this set, we add $(AND\ P_1 \ldots P_i \ldots)$ $\forall P_i$ in CP_C
In the same way, to every role R_i not occurring in C $(NOTInd(R_i))$ is added.

The idea is that after the processing of a definition, the only concepts and roles permitted for these concrete concepts are accompanied by their indicator. If the coherency between a concrete concept C_c and a conceptC_1 is to be proved, we have:

if R is not used in C_c, $(NOTInd(R))$ is in the definition (after transformation) of C_c, if the same role R is used by (or inherited by) concept C_1, $Ind(R)$ will be present in its definitions. Finally leading to a clash, sign of inconsistency. The same for concepts: if C is not used in C_c, $(NOTInd(C))$ is in its definition etc.

Example:

```
(define-primitive-concept module TOP)
(define-primitive-concept controler module)
(define-primitive-concept rollover-sensor
module)
(define-primitive-concept crash-sensor module)
(define-primitive-concept seat-controler
controler)
(define-concept combi-sensor
(AND rollover-sensor crash-sensor))
```

`combi-sensor` and `seat-controler` are concrete concepts.

Suppose we define now a concept `combi-module` :

```
(define-concept combi-module
(AND seat-controler combi-sensor)
```

after replacements and after applying the meta-constraint the concept `combi-module` becomes:

```
(AND
(AND (AND seat-controler (NOT Ind(module))
(NOT combi-sensor))
(AND (AND combi-sensor Ind(module))
(NOT seat-controler)))
```

The tableaux calculus algorithm for this closed TBox will lead to a clash, because the closure of both concrete concepts `seat-controler` and `combi-sensor` has introduced conflicting indicators. In expanding `seat-controler`, `(NOT Ind(combi-sensor))` has been introduced to exclude the other concrete concepts. Expanding `combi-sensor` will introduce `combi-sensor` and cause a clash. The next example shows how the process of introducing Indicator and Not Indicator can be recursively applied on every part of disjunction found in a concept definition so the process is not restricted to "structural" concept definition i.e. definition excluding ORs.

This example gives some idea about the bus application

```
(define-primitive-concept module TOP)
(define-primitive-concept bus TOP)
(define-primitive-concept controler module)
(define-primitive-concept sensor module)
(define-primitive-concept Steller module)
(define-primitive-role has-element R)
(define-primitive-role has-component R)
(define-concept diagnostic-bus
(AND Bus (SOME has-element sensor)))
(define-concept Stellbus
(AND Bus (SOME has-element Steller)))
(define-concept combi-bus
(AND Stellbus diagnostic-bus))
(define-concept bussystem
( AND controler (SOME has-component
diagnostic-bus)
(SOME has-component Stellbus)))
```

after replacements the abstract concept `sensor` and the concrete concept `diagnostic-bus` are:

sensor:

```
(AND module Ind(module))
```

diagnostic-bus

```
(AND
(AND bus Ind(bus))
(AND(AND SOME has-element)
Ind(has-element))
(AND sensor Ind (sensor))
(NOT Ind(Steller)) (NOT Ind(controler))
(NOT Ind(module)) )
```

It is easy to see that (AND sensor diagnostic-bus) causes a clash coming from the contradiction between `Ind(module)` and `(NOT Ind(module))`.

4 Configuration process by instance refinement

In this section, instance checking must be introduced. The problem is to have a way of verifying the consistency of a Knowledge Base i.e. of the couple TBox ABox. First, to compute the consistency of a couple instance, concept, it is sufficient to prove that the intersection between the instance and the extension of the concept is not empty. It is made by using the tableaux algorithm. Given I an instance, its description I_d is defined by a conjunction of assertions.

If C is a concept to examine, we consider the expression $And\ I_d\ C$. The constraints corresponding to this expression are expanded in the constraint set of tableaux. If a completion without clash is found, it means that the intersection is not empty.

In order to verify the coherency of one couple TBox, ABox , all constraints corresponding to assertions and corresponding to concepts expressions are examined and if every completion causes a clash, it can be deduced that the couple is inconsistant.

The configuration process is based on the computing of some interesting sets of concept.

Suppose for the moment that each disjunctive expression corresponding to a concrete concept has only one conjunctive argument.

The set $POSS$ of possible concepts is the set of the concepts that the instance can possibly instanciate. To calculate the $POSS$ related to an instance I, the concepts for which the intersection between the instance and the extension of the concept is not empty are computed.

The set $SPOSS$ of the smallest possible concepts is calcuted from the previous one using susbsumption. Theses concepts can help in providing more accurate possibilities. As a matter of fact, it is more interesting for the system to propose to the user more specific choices.

The set IMP of impossible concepts is the complementary of the previous one.

The set $BIMP$ of the biggest impossible concepts is calculated from the previous one. It is used in a dual manner as $SPOSS$.

The set NEC of necessary concepts contains the concepts of which I can be an instance. The tableaux algorithm is used to prove that $C \Rightarrow I_d$ i.e. It is to be shown that the set of constraints corresponding to And I_d $(NOTC)$ produces always a clash.

The configuration process

First the instance I is completely unspecified : $I \in \top$ is the only assertion defining I. The set of possible concepts $POSS$ is the set of all the concepts of the terminology, the set NEC of necessary concepts is equal to $\{\top\}$, the set IMP of impossible concepts is empty.

After adding one ore more assertions to refine I, we have to test the consistency of the couple terminology-instance. If there is no inconsistency, the new sets are to be calculated. The new NEC is obtained by doing in-depth processing, i.e. by scanning every descendant of the concepts of the previous NEC: it is included in the preceding one. All necessary concepts are possible. So the new $POSS$ set is based on the old $POSS$ set by examining only the concepts which are not in NEC.

The cycle sketched here goes on until the sets cannot be refined by adding assertions. It is possible to compute at each step which assertions can be added to the instances: assertions concerning primitive concepts or concerning roles. So possible choices and only possible choices can be proposed via an interface. The consequences of choices done are then automatically taken into account and propagated.

In the general case where some ORs can occur at the first level: the system should help exploring every alternative opened by an OR branch.

The present implementation works manually. It is used to experiment configuration problems in the automotive industry.

An interface is to be built: the problem is to conceive this interface to be simple enough to help the user in first completing its instance second to explore alternatives if there are some.

The system presently provides a help to configure a bus, called "CAN bus", used in the automotive domain. This bus connects all the electronic elements: sensors, captors and security gear needed for the security in a car. The problem is to find devices devoted to certain security functions and find locations for these devices in a car. Devices are partially devoted to control(controlers), partially devoted to detection (sensors) and to protection (ABS, airbag,etc.). Constraints are of different kinds : some are spatial, some are device dependant: a controler must be put in a location close to its device, etc.

5 Some improvements

Notice that main improvements consist in the use of multi-TBoxes and the use of a constraint solver on attribute's value. A modulee specialized in the resolution of numerical constraints (concrete domains) is used as already seen. Numerical constraints can be put on values of attributes (role with a unique value), constraints can only be linear. As alreday seen, the numerical concrete domain is usefull to cope with spatial constraints. But the idea to separate a TBox into several smaller ones can be generalized. Using several TBoxes helps in reducing general complexity and is a good way to be modulear. The TBoxes can communicate by the way of special roles declared as connectors. The condition is that the TBoxes are independent.

In the bus configuration application, we use TBoxes for concrete domains were problems are solved independently of problems of the main terminology [6]. Roles connecting Tboxes are: between functions and modulees "is-realized-by" and its inverse role "realizes", between modulees and busses connects and is-connected-to, between events and functions "ask-for" "respond-to", between events and busses "is-transmitted-by", "transmits". This partitionning has the advantage of separating clearly the different kinds of knowledge needed to solve our configuration problem. It facilitates the modelling by giving a homogeneous way to let the different parts communicate. The second important advantage is indeed, the gain in computitational complexity: it is sure that having small terminologies provides more efficiency.

The present prototype shows good capabilities to solve this particular configuration task. It remains to test it on bigger KB and on other problems in order to develop a generic system able to solve general configurations tasks.

6 Conclusion

DL appears to be a good tool in helping configuration process, because it allows the declarative specification of the set of constraints needed to solve the problem. It is clear that this ability to manipulate constraints and to reason under constraints is interesting for tasks that are not strictly classified under the label of configuration. We think that this extension is well adapted to the management of dynamics forms, and to the control of the coherency of orders etc.

References

[1] Horrocks I., 'Optimization Techniques for Expressive Description Logics', Department of Computer Science,University of Manchester.1997. Technical report Series UMC-97-2-1.

[2] Wright J., Weixelbaum E., Brown K.,Vesonder G., Palmer S., Berman J., Moore H., 'A knowledge - based configurator that supports Sales, Engineering and manufacturing at AT&T network systems' Proceedings of the Innovative Applications of Artificial Intelligence Conference pp183-193,1993.

[3] Weida R., 'Closed Terminologies and Temporal Reasoning in Description Logic for Concept and Plan Recognition' Ph.D. Dissertation,1996 Columbia University, USA.

[4] Compatangelo E., Donini F.M., Rumolo G. 'DL-based Support to Domain Engineering' Proc of DL' 98 Trente.

[5] Peter F. Pattel-Schneider 'DL Systems comparison'. Proc of DL'99

[6] F. de Beuvron 'Presentation of the DL system CICLOP' Proc of DL'99

[7] Moeller R., 'Reasoning About Domain Knowledge and User Actions for interactive Systems Development' in Proc IFIP Working Groups 8.1/ 13.2 Conference Domain Knowledge for Interactive System Design, Geneva 8-10th May Chapman &Hall Publishers 1996.