Modeling and Verifying Complex Objects : A Declarative Approach Based on Description Logics

Marie-Christine Rousset and **Pascale Hors** [1]

Abstract.
Description logics are good candidates for representing complex objects. Their expressive and computational properties have been extensively studied and are well understood. However, from a knowledge engineering perspective, the current reasoning services provided by description logic systems do not suffice for representing objects. Adding services (e.g help for debugging) is necessary to make description logic systems useful tools for building object-centered knowledge bases. In this paper, we present a principled *glass-box* approach both for trace-based debugging and for supporting transparent modeling of whole-part relationship, using description logics. This approach has been proven very useful for a real application of modeling components of power plants.

1 Introduction

Among the numerous existing object-centered formalisms, description logics are good candidates to represent complex objects. They are declarative and they are based on a clear semantics while associated with algorithms that provide classification services. The computational and expressive properties of description logics have been extensively studied. Several systems have been built based on description logics (e.g., CLASSIC [3], LOOM [8], BACK [12], KRIS [1]), they have been used in several applications (e.g., [16, 14]). A description logic knowledge base includes descriptions of *concepts* representing classes of objects in the domain. A concept is defined by the conditions that must be satisfied by elements in the class. A description logic contains a set of *constructors* that allows to define these conditions at various levels of complexity. Much of the research in description logics has concentrated on algorithms for determining subsumption relations between classes, and for checking membership of an object in a class.

Description logics have good representational properties and their computational properties are well understood. However, from a knowledge engineering perspective, the current reasoning services provided by description logic systems do not suffice for representing objects. Additional services are necessary to make description logic systems useful tools for building object-centered knowledge bases.

In particular, like for any other software system, debugging services are necessary for supporting building object-oriented knowledge bases. The problem with description logic systems is that their current reasoning algorithms have an inherent black-box nature, i.e they don't make explicit how inferences are obtained. Consequently, they are ill adapted for providing inference traces, and then debugging remains an open question.

Furthermore, the possible distinctions that are meaningful from a modeling point of view are not necessarily preserved at the knowledge representation level. In particular, describing objects in terms of their components is central in modeling physical devices. However, very few attempts have been done to *transparently* represent the underlying whole-part relation in description logics.

In this paper, we present a principled *glass-box* approach both for trace-based debugging and for supporting transparent modeling of whole-part relationship, using description logics. We have based our study on a subset of description logic constructors that we have found specially useful in a real application of modeling components of power plants [5] [2]. We have then showed how to customize our approach in order to combine it with specific description logic systems like CLASSIC or BACK.

First, we describe our glass-box consistency checking method. In addition to detecting inconsistencies, it provides a trace of some of the subsumption steps leading to the inconsistency, in the form of a *trace-tree*. A trace-tree is built from a set of *trace rules*, each of them making explicit a schema of subsumption that is informative for debugging purposes. Our trace rules summarize a certain number of complex subsumption relationships that cannot be reduced to structural or superficial subsumption checking.

Then, we describe our approach for representing the whole-part relation in a *transparent* way, i.e, which makes possible that this specific relation and the inferences that can be drawn through it are explicitly distinguished. Our approach is based on expressing the whole-part relation with a specific role *part* and on forcing its transitivity by means of rules that are used to complete descriptions involving whole-part relations. In addition, these rules can be used to provide traces of inferences in which the whole-part based inferences are distinguished from the other subsumption inferences.

2 Description Logics

A description logic contains unary relations (called concepts) which represent classes of objects in the domain and binary relations (called roles) which describe relationships between objects. Concepts and roles can be either primitive or complex. Complex concepts and roles are defined via *descriptions* built from a set of constructors that varies from one description logic to another. We present a certain number of standard constructors the expressive power of which have been proven appropriate in a real application of modeling components of nuclear plants ([5]). We use the following syntax (A denotes a primitive concept, C and D represent arbitrary concepts, P denotes a primitive role and R represents an arbitrary role):

[1] L.R.I. U.R.A C.N.R.S, University of Paris-Sud, mcr@lri.lri.fr

[2] this work have been funded by Electricite de France

ECAI 96. *12th European Conference on Artificial Intelligence*
Edited by W. Wahlster
Published in 1996 by John Wiley & Sons, Ltd.

C,D →	A |	(primitive concept)
	$\top$ | $\bot$ |	(top, bottom)
	C $\sqcap$ D |	(conjunction)
	not A |	(complement of primitive concepts)
	(all R C) |	(all restriction)
	(at-least n R) |	(at-least restriction)
	(at-most n R)	(at-most restriction)
R →	P |	(primitive role)
	(P and range C)	(range restriction)

The semantics of concept and role expressions is given via *interpretations*. An interpretation I contains a non-empty domain $\mathcal{O}^I$. It assigns an unary relation C^I to every concept, and a binary relation R^I over $\mathcal{O}^I \times \mathcal{O}^I$ to every role R. The extensions of the complex concepts and roles satisfy the following equations ($\sharp\{\}$ denotes the cardinality of a set):

$\top^I = \mathcal{O}^I$, $\bot^I = \emptyset$, $(C \sqcap D)^I = C^I \cap D^I$, (not A)^I = $\mathcal{O}^I \setminus A^I$,
(all R C)^I = $\{d \in \mathcal{O}^I \mid \forall e : (d, e) \in R^I \rightarrow e \in C^I\}$
(at-least n R)^I = $\{d \in \mathcal{O}^I \mid \sharp\{e \mid (d, e) \in R^I\} \geq n\}$
(at-most n R)^I = $\{d \in \mathcal{O}^I \mid \sharp\{e \mid (d, e) \in R^I\} \leq n\}$
(R and range C)^I = $\{(d1, d2) \mid (d1, d2) \in R^I \wedge d2 \in C^I\}$

A terminology is a set of *definitions* and *inclusion statements*. A concept (respectively role) definition is a statement of the form $A := D$, where A is a concept (role) name and D a concept (role) description. We assume that a concept (role) name appears on the left hand side of atmost one definition statement. An inclusion statement is of the form $C \sqsubseteq D$, where C and D are concepts (role) descriptions. Intuitively, a concept definition associates a concept description to a concept name, and a concept inclusion states that every instance of the concept C must be an instance of D. A concept name A is said to *depend* on a concept name B if there is some statement in the terminology in which A appears in the left hand side and B appears in the right hand side. The terminology is said to be *cyclic* if there is a cycle in the dependency relation.

Definition 2.1: *An interpretation I is a model of a terminology $\mathcal{T}$ if $C^I \subseteq D^I$ for every inclusion statement $C \sqsubseteq D$, and $A^I = D^I$ for every definition statement $A := D$.*

In this paper, we restrict the inclusions statements to express only hierarchies on primitive concepts or roles and disjointness between primitive concepts. Furthermore, we only consider terminologies with no cycle. As an example of a terminology, consider the following $\mathcal{T}_1$:

InternalSide $\sqsubseteq$ ImpellerSide
ExternalSide $\sqsubseteq$ ImpellerSide
InternalSide $\sqcap$ ExternalSide $\sqsubseteq \bot$
Internal-Impeller:= (at-least 2 (part and range InternalSide))
External-Impeller:= (at-least 2 (part and range ExternalSide))
VerticalSide:= (all (part and range ImpellerSide) Vertical)
StaticPart := (at-most 3 (part and range Mobile))
ComplexComp := (at-least 4 part)
MobilePart := (at-most 1 (part and range (not Mobile)))

InternalSide, ExternalSide, ImpellerSide, Vertical and Mobile are primitive concepts, and part is a primitive role. A Internal-Impeller (respectively, External-Impeller) is defined to be the set of elements that have at least 2 parts that are internal sides (resp., external sides). The concept VerticalSide represents the elements whose all the parts that are impeller sides are vertical. The concept StaticPart (resp., MobilePart) represents the elements that have at most 3 parts which are mobile (resp., atmost 1 part which is not mobile). The concept ComplexComp represents the elements that have at least 4 parts.

The subsumption relationship between two concepts corresponds to the "is-a" relationship and is defined based on formal semantics.

Definition 2.2: *C is subsumed by D w.r.t. $\mathcal{T}$ (noted $C \preceq_{\mathcal{T}} D$) if $C^I \subseteq D^I$ in every model I of $\mathcal{T}$.*

As an example of subsumption relationship that can be logically infered from the terminology $\mathcal{T}_1$, the concept Internal-Impeller $\sqcap$ External-Impeller is subsumed by ComplexComp. This follows because instances of first description must have at least 2 parts that are internal sides and at least 2 that are external sides, and since InternalSide and ExternalSide are disjoint, they must have at least 4 parts, which, by definition, means being a ComplexComp.

Much of the research in description logics has concentrated on algorithms for determining subsumption relations between concepts, and for checking membership of an object in a concept. In particular, theoretical results ([10]) show that the range restriction for roles combined with the number restriction constructors cause subsumption checking to be intractable in the worst cases. In parallel, several systems have been built based on description logics (e.g., CLASSIC [3], LOOM [8], BACK [12], KRIS [1]). Specific description systems do not necessarily offer all the constructors that are considered in this paper. For instance, CLASSIC does not include the range restriction constructor for roles. The counterpart of its restricted language is that the subsumption algorithm implemented in CLASSIC is complete and polynomial. In contrast, for other description systems like BACK the choice has been made to offer a more expressive language (including all the constructors that have been presented here) but to offer only *incomplete* subsumption inference services for efficiency reasons.

3 A Glass-Box Approach for Terminology Consistency Checking

Description logics provide a logical setting allowing one to define simply and rigorously consistency of a terminology. Guaranteeing the existence of a model for the terminology is not sufficient for a modeling and debugging point of view. What is needed in addition is to determine whether *any concept description* involved in concept definitions represents a non-empty concept. For doing so, since the terminologies that we consider have no cycles, we *unfold* the part of the terminology composed of concept definitions by iteratively substituting every concept name with its definition. As a result, we obtain an unfolded terminology (denoted unfold(T)) composed of a set of inclusions statements involving only primitive concepts or roles (denoted incl(T)) and a set of concept definitions that associate a concept name with a description in which only names of primitive concepts and roles appear.

Definition 3.1: *A terminology T is consistent iff all the concept descriptions appearing in definitions of unfold(T) are consistent. A concept description D of unfold(T) is inconsistent iff $D \preceq_{incl(\mathcal{T})} \bot$.*

Theoretical research on subsumption ([10]) or satisfiability problems ([4]) provide results about complexity of terminology consistency checking, and subsumption algorithms implemented in existing description systems can be used to effectively check consistency of a terminology. However, for debugging purposes, consistency checking is not an end in itself. Detecting an inconsistency is the starting point of a debugging process that aims at helping the knowledge base designer to find the cause of that inconsistency. The problem is that the existing subsumption algorithms, by their black-box nature, cannot provide trace services that are meaningful for the knowledge base designer.

The glass-box approach that we propose can be used as a complement of existing subsumption algorithms, both to possibly complete existing subsumption algorithms and to provide traces for the most subtle subsumption inferences. The basic assumption of our approach is that the subsumption relations that are the most difficult to understand for a terminology designer are the subsumption relations that are not structural, in particular those stating that a conjunction of concepts is subsumed by a concept that does not not subsume any of the conjuncts. Our approach consists of making explicit some schemas of subsumption relations that are not structural as *trace rules*. These trace rules are then used to build *trace-trees* that explain how such steps of non structural subsumption can be chained to result in a complex subsumption relation.

Our approach has debugging purposes and is guided by theoretical results that are summarized in the following theorems 1 and 2. Theorem 1 suggests that if the global process of checking consistency for a terminology is performed according to the increasing level of depth of descriptions, only conjunctions of concepts have to be checked.

Theorem 2 suggests that any complex proof of inconsistency of a conjunction, can be splitted in intermediate simpler steps of subsumption or inconsistency checking. Our glass-box approach is based on this theorem, since it relies on identifying some steps of subsumption as trace-rules, in order to build trace-trees that are proofs chaining steps of subsumption.

Definition 3.2: *Let D a concept description. D is said complex if it is a conjunction. It is said simple otherwise.*
A complex concept description D is then the conjunction of a certain number of simple conjuncts $F_1 \sqcap \ldots \sqcap F_n$, which are called its subterms. The depth of D is $1 + Max\{depth(F_1) \ldots depth(F_n)\}$

Definition 3.3: *Let D a simple concept description. Its subterms and its depth are defined according to its form as follows (P denotes a primitive role, A a primitive concept different of $\top$ and $\bot$, F and C concept descriptions).*
- *If D is either a description of the form (at-least n P) or (at-most n P), or of the form A or not A, it has no subterm, and its depth is 0.*
- *If D is a description of the form (all P C) or (at-least n (P and range C)) or (at-most n (P and range C)), its subterm is C and its depth is $1 + depth(C)$.*
- *If D is a description of the form (all (P and range F) C), its subterms are F and C, and its depth is $1 + Max\{depth(F), depth(C)\}$*

For example, (at-least 2 (part and range InternalSide)) $\sqcap$ (all (part and range ImpellerSide) Vertical) is a complex description concept. Its subterms are the simple concepts (at-least 2 (part and range InternalSide)) and (all (part and range ImpellerSide) Vertical). The subterms of the second conjunct are ImpellerSide and Vertical.

Theorem 1 *Let D a* simple *concept description of an unfolded terminology. If D has no subterm, or if its subterms are consistent, then it is consistent.*

The proof follows by induction and consists in building a model for any form of simple concept description, from an existing model of its subterms.

We perform the global process of consistency checking of a terminology by checking consistency of each of its concept descriptions by following the order of increasing depth. In doing so, when the process reaches a description concept D, its subterms have been checked previously and they are consistent (otherwise the process would have stopped before reaching D). Based on theorem 1, if D is simple, it is consistent. The process can then focus on checking consistency on complex concept description.

Theorem 2 *Let $D : D_1 \sqcap \ldots \sqcap D_n$ a complex concept description that has n simple and consistent conjuncts.*
$D_1 \sqcap \ldots \sqcap D_n \preceq \bot$ iff either $\exists$ i,j s.t $D_i \sqcap D_j \preceq \bot$ or there exists k simple and consistent concepts $C_1, \ldots C_k$ such that :
- *$k < n$,*
- *$C_1 \sqcap \ldots \sqcap C_k \preceq \bot$,*
- *$\forall i \in [1..k], \exists l_1 \ldots l_i \in [1..n] s.t D_{l_1} \sqcap \ldots \sqcap D_{l_i} \preceq C_i$*

The proof is a proof by induction on n. It uses a result proved in [6] about disjointness about a simple concept description and a complex concept description depending on the form of the simple concept.

Our glass-box approach is founded on theorem 2 and consists of accounting for a certain number of inferences of the form $C_1 \sqcap \ldots \sqcap C_i \preceq C$ by trace rules. A trace rule has the form

$$PC_1, \ldots, PC_k, IC_1, \ldots, IC_l \Rightarrow PC$$

where $PC_1, \ldots, PC_k$ are patterns of simple concepts, and $IC_1, \ldots, IC_l$ are interpreted conditions on variables appearing in the patterns PC_i. A pattern of a simple concept is a concept description, some subterms of which can be variables. The interpreted conditions that we consider can be arithmetic comparisons about some cardinality parameters, and also subsumption checking or disjointness checking between some variables representing concept or role descriptions. As an example, consider the following trace rules (the variables are in capital letters).

Example 3.1:

r_1:(at-least K_1 (R and range C_1)), (at-least K_2 (R and range C_2)),
$C_1 \sqcap C_2 \preceq \bot, C_1 \preceq C, C_2 \preceq C, K = K_1 + K_2$
$\Rightarrow$ (at-least K (R and range C)
r_2:(at-least K_1 (R and range C)), (at-least K_2 (R and range not C)),
$K = K_1 + K_2 \Rightarrow$ (at-least K R)
r_3: (at-most K_1 (R and range C)), (at-most K_2 (R and range not C)),
K = K_1+K_2 $\Rightarrow$ (at-most K R)
r_4:(at-least K_1 (R and range C_1)), (all (R and range C_2) C),
$C_1 \preceq C_2 \Rightarrow$ (at-least K (R and range C))
r_5: (at-least K_1 R), (at-most K_2 R), $K_2 < K_1 \Rightarrow \bot$
r_6: (at-least K_1 R_1), (all R_1 C_1), (at-least K_2 R_2), (all R_2 C_2),
$C_1 \sqcap C_2 \preceq \bot, R_1 \preceq R, R_2 \preceq R, K = K_1 + K_2 \Rightarrow$ (at-least K R)
r_7: (at-most K_1 R_1), (all R_1 C), (at-most K_2 R_2). (all R_2 not C),
$R_1 \preceq R, R_2 \preceq R, K = K_1 + K_2 \Rightarrow$ (at-most K R)

Each trace rule makes explicit a non structural subsumption schema between a complex concept description and a simple one. The pattern conditions of the rule represent the simple conjuncts of the complex concept, and its conclusion represents the subsuming simple concept. Its interpreted conditions express some constraints that have to be satisfied for the schema subsumption to hold.

Property 3.1: Given a trace rule $PC_1, \ldots, PC_k, EC_1, \ldots, EC_l \Rightarrow PC$. For any mapping ϕ from the variables appearing in the rule to constants such that $\phi(EC_1) \wedge \ldots \wedge \phi(EC_l)$:

$$\phi(PC_1) \sqcap \ldots \sqcap \phi PC_k \preceq \phi(PC), \ \forall i \in [1..k], \phi(PC_i) \not\preceq \phi(PC)$$

For example, the instance of r_1 corresponding to the mapping $\phi = \{K_1 \rightarrow 2, K_2 \rightarrow 2, K \rightarrow 4, R \rightarrow part\ C_1 \rightarrow$ InternalSide, $C_2 \rightarrow$ ExternalSide$\}$ supports the subsumption relation : (at-least 2 (part and range InternalSide)) $\sqcap$ (at-least 2 (part and range ExternalSide)) $\preceq$ (at-least 4 part).

A trace rule represents a one-step not structural subsumption. A trace tree summarizes different steps of subsumption, which, when they are chained, account for a complex subsumption.

Definition 3.4: *A trace tree is a tree that has two types of nodes, concept-nodes, which represent simple concepts, and rule-nodes(R, ϕ), which represent instances of trace rules. A rule-node (R, ϕ) is labelled by the mapping ϕ of its interpreted conditions. A concept-node CN is either a leaf or it has a rule-node child that represents a trace rule whose the consequent can be unified with CN. A rule-node (R,ϕ) has a concept-node child $\phi(PC_i)$ for each of its pattern conditions PC_i.*

Definition 3.5: *A trace of a subsumption $C_1 \sqcap \ldots \sqcap C_k \preceq C$, where $C_1, \ldots, C_k, C$ are simple concepts is a trace tree whose root is C, and whose leaves are and $C_1, \ldots, C_k$.*

Note that given a subsumption query and its trace in the form of a trace-tree, the designer can ask for traces for other subsumption relations, and in particular, for the subsumption relations appearing in labels of rule nodes of the trace-tree.

An example of a trace tree providing the trace of an inconsistency is given in the figure1. It occurs when it is proposed to add to the terminology $\mathcal{T}_1$, a concept defined as the conjunction of Internal-Impeller, External-Impeller, VerticalSide, MobilePart, StaticPart and (at-least 1 (part and range not Vertical)).

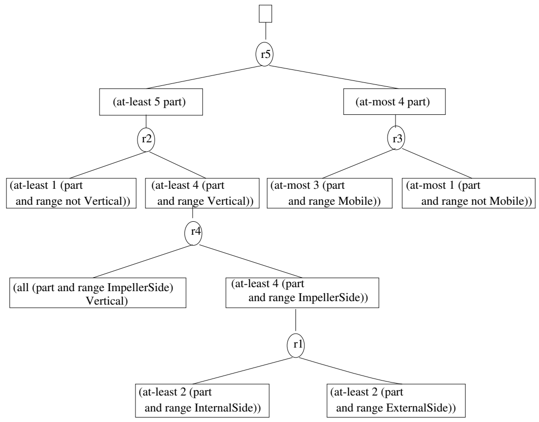

Figure 1. An example of a trace-tree : for clarity reasons, the labels of the rules are not shown

Given a a set of trace rules, and a subsumption query $C_1 \sqcap \ldots \sqcap C_k \preceq C$, building a trace tree can be done by applying any standard (forward chaining or backward chaining) inference procedure for rules. We have implemented a forward chaining procedure in Prolog.

Currently, we have a set of trace rules that contains about 15 rules with variables. Though they are not complete, they cover many cases of not structural subsumption relations about conjunctions of simple concepts. In particular, they were very useful to detect complex and hidden inconsistencies of our application on modeling components of power plants ([5]). Each rule expresses a non structural subsumption resulting from a specific combination of certain form of simple concepts. In this way, our method can be easily customized to any existing description system whose constructors are a subset of the constructors that we considered, by restricting the set of trace rules to an appropriate subset. For example, for using our method to CLASSIC terminologies, the trace rules r_1, r_2, r_3 and r_4 are removed.

When applied in association with a description logics system providing a complete subsumption algorithm (e.g CLASSIC), our rules are just useful for trace purposes. When applied in association with a description logics system with an incomplete subsumption algorithm, they can also be useful to make the subsumption checking more complete. In particular, our set of rules provides a way to detect a lot of subsumptions which are not detected by the subsumption algorithm associated with the system BACK.

4 Transparently representing whole-part relation in a description logics setting

In many real world applications, composed objects play a central role. Various types of whole-part relations have been pointed out ([15]), underlying different kinds of composed objects. In this paper, we focus on modeling and reasoning aspects of the physical whole-part relation, which accounts for physically composed products or devices. Several types of inferences through whole-part relations can be considered, some of them are specific to some properties in some domains (e.g, the colour or the cost of a whole based on colour or costs of its part), some others are generic. In this paper, we consider a minimal set of generic inferences, which states how whole-part relations interact with is-a relations, and also the transitivity of the whole part relation. They are summarized by the three following schemas of inference, where A $n \leq part \leq m$ B denotes that any instance of the class (i.e, concept) A has at least n and at most m components that are instances of the class B.

A $n \leq part \leq m$ B , C $\preceq$ A $\vdash$ C $n \leq part \leq m$ B
A $n \leq part \leq m$ B , B $\preceq$ C $\vdash$ A $n \leq part$ C
A $n \leq part \leq m$ B , B $n' \leq part \leq m'$ C $\vdash$ A $n' \leq part \leq m * m'$ C

Expressing whole-part relation by a specific role *part* is straightforward in a description logics language where the range restriction constructor is available. A $n \leq part \leq m$ B is then directly expressed by :
A $\preceq$ at-least(n, part and range(B)) $\sqcap$ at-most(m, part and range(B)).

It is possible to compensate for the lack of the range restriction constructor and still being able to appropriately express the whole-part relation in more restricted languages like CLASSIC by creating as many primitive roles $part_B$ as classes of components B, and by adding in the terminology as many primitive role inclusion statements $part_B \sqsubseteq part_C$ as there exist subsumption relations $B \preceq C$.

In both cases, the two first schemas of inference are completely captured by subsumption inference.

The third inference schema expresses the property of transitivity of the whole-part relation and the way it conveys the cardinality constraints on the number of parts. It requires a specific implementation simulating transitivity inferences on the specific role(s) expressing the whole-part relation. The solution we propose consists of expressing the transitivity properties of the whole-part relation by rules, which are similar in the form to trace rules, but which are used as completion rules that explicitly add conjuncts to concept descriptions accounting for transitivity of the part role. Examples of completion rules are :

cr_1 : (at-most K_1 (part and range B)),
B $\preceq$ (at-most K_2 ((part and range C)), K = $K_1 + K_2$
$\Rightarrow$ (at-most K (part and range C))
cr_2 : (at-least K_1 (part and range B)),
B $\preceq$ (at-least K_2 ((part and range C))
$\Rightarrow$ (at-least K_2 (part and range C))

A query asking about whole-part relations corresponds to a subsumption checking query $D \preceq G$, where G is a simple concept of the form *(at-least n (part and range F))* or *(at-most n (part and range*

F)), and D is a conjunction of simple concepts $F_1 \sqcap \ldots F_n$. Such a query is processed in two phases.

The first phase consists of applying the completion rules on the set of conjuncts of D, resulting in a *completed* description of D (noted compl(D)), which accounts for transitivity property of the part role. It is checked whether the mapping of the variable C in consequent of the rules to the concept description F appearing in the query makes the antecedents of one of the rules to be satisfied w.r.t the set of conjuncts of D. If that is the case new conjuncts are added to the initial description D accounting for one step of the transitivity inference. The completion rules are applied until saturation, i.e until no new instance is satisfied.

The second phase consists of checking whether $compl(D) \preceq G$. The completion rules together with the trace rules, presented in the previous section, in which the role variables are instantiated with the specific role *part*, can be also used for providing a trace of inferences both of the completion phase and of the subsumption checking $compl(D) \preceq G$.

5 Discussion

In this paper we presented a principled glass-box approach for modeling and verifying complex objects, which aims to complement existing reasoning algorithms in a description logics setting. Description logics have good representational properties that make them powerful tools for representing complex objects. On the other hand, the inherent black-box nature of the inference algorithms that are currently associated with existing description logics systems is the main drawback to use them in a modeling and knowledge engineering purpose. Our approach is based on making explicit as rules some steps of subsumption inferences that are meaningful for a debugging or modeling purpose. Those rules can then be chained to provide traces of more complex inferences. In addition, when combined with description systems that do not provide a complete subsumption algorithm, our approach is useful to make subsumption checking more complete. A real application of modeling components of nuclear plants showed the usefulness of such an approach for helping to design a complex object-centered knowledge base in association both with the CLASSIC and BACK description systems.

Our work can be compared with several existing works on different aspects. In [9], the notion of explanation is specified for presenting subsumption inferences of the CLASSIC system. This work is quite similar in spirit to ours because relying on explicit rules for accounting for some subsumption inferences. The main difference is that in [9], the only explanations that are provided are traces of structural and then rather simple subsumption relations, though our approach focusses on providing traces of more complex subsumptions that *cannot* be reduced to structural subsumptions. Furthermore, the description logic that we consider is much more expressive than the one underlying the CLASSIC system.

Some other works have considered the problem of representing whole-part relations in a description logics setting. In [13], an extension of the description system CLASSIC is proposed to handle some aspects of the physical whole-part relation. In particular, the transitivity of whole-part relation is not implemented. In [11], a theoretical framework based on description logics is presented to define compositional inferencing as a process which infers the existence of a whole, based on the existence of the required parts. Our approach differs from both of these works in that it focuses on the interaction between subsumption and whole-part based inferences and that it makes this interaction explicit for the knowledge base user or designer.

More generally, the use of rules for describing subsumption inference can be related to a proof-theoretic approach for defining subsumption. Different authors ([2], [7]) have advocated the advantages of a proof-theoretic approach for characterizing or customizing implementations of subsumption algorithms. Our work shows that a proof-theoretic approach is crucial for debugging purposes. It has to be noticed however that the rules that are meaningful for debugging purposes are not necessarily the inference rules that are appropriate or relevant for other purposes.

ACKNOWLEDGEMENTS

We thank Alon Levy for discussions and comments on earlier versions of this paper, and Marc Porcheron for his help on the example.

REFERENCES

[1] F. Baader and B. Hollunder, 'A terminological knowledge representation system with complete inference algorithm', in *In Proceedings of the Workshop on Processing Declarative Knowledge, PDK-91, Lecture Notes in Artificial Intelligence*, pp. 67–86. Springer-Verlag, (1991).

[2] Alexander Borgida, 'From type systems to knowledge representation : Natural semantics specifications for description logics', *Int. Journal of Intelligent and Cooperative Information Systems*, (1992).

[3] R. J. Brachman, A. Borgida, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Resnick, 'Living with CLASSIC: When and how to use a KL-ONE-like language', in *Principles of Semantic Networks*, ed., John Sowa, 401–456, Morgan Kaufmann, San Mateo, CA, (1991).

[4] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf, 'Decidable reasoning in terminological knowledge representation systems', *Journal of Artificial Intelligence Research*, **1**, 109–138, (1993).

[5] Pascale Hors, *Representation et Verification de Connaissances de Structuration a l'aide de Systemes Terminologiques*, Ph.D. dissertation, Universite Paris-Sud, Orsay, France, 1995.

[6] Pascale Hors and Marie-Christine Rousset, 'Consistency of structured knowledge : a formal framework based on description logics.', *International Journal of Expert Systems and Applications*, **8(3)**, (1995).

[7] J.Quantz and V. Royer, 'Deriving inference rules for terminological logics', in *Proceedings of JELIA 92*, (1992).

[8] R. M. MacGregor, 'A deductive pattern matcher', in *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 403–408, (1988).

[9] Deborah L. McGuinness and Alexander T. Borgida, 'Explaining subsumption in description logics', in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, (1995).

[10] B. Nebel, 'Terminological reasoning is inherently intractable', *Artificial Intelligence*, **43(2)**, 235–249, (1990).

[11] Lin Padgham and Peter F. Patrick Lambrix, 'A framework for part-of hierarchies in terminological logics', in *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, (1994).

[12] C. Petalson, 'The BACK system : an overview', in *Proceedings of the SIGART bulletin*, volume 2(3), pp. 114–119, (1991).

[13] Piet-Hein Speel and Peter F. Patel-Schneider, 'Classic extended with physical whole-part relations', in *Proceedings of the International Workshop on Description Logics, Bonn, Germany*, (1994).

[14] W. Wahlster, E. Andre, W. Finkler, H.J. Profitlich, and T. Rist, 'Plan-based integration of natural language and graphics generation', *Artificial Intelligence*, **63(1-2)**, 387–428, (1993).

[15] Morton E. Winston, Roger Chaffin, and Douglas Herrmann, 'A taxonomy of part-whole relations', *Cognitive Science*, **19**, (1987).

[16] J.R Wright, E.S. Weixelbaum, K. Brown, G.T Vesonder, S.R Palmer, J.I Berman, and H.H Moore, 'A knowledge-based configurator that supports sales, engineering and manufacturing at at&t network systems', in *Proceedings of the Conference on Innnovative Applications of Artificial Intelligence Conference*, pp. 183–193, (1993).