Terminological Systems Revisited: Terminology = Schema + Views*

M. Buchheit[1] and **F. M. Donini**[2] and **W. Nutt**[1] and **A. Schaerf**[2]
1. German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
{buchheit,nutt}@dfki.uni-sb.de
2. Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Italy
{donini,aschaerf}@assi.dis.uniroma1.it

Abstract

Traditionally, the core of a Terminological Knowledge Representation System (TKRS) consists of a so-called TBox, where concepts are introduced, and an ABox, where facts about individuals are stated in terms of these concepts. This design has a drawback because in most applications the TBox has to meet two functions at a time: on the one hand, similar to a database schema, framelike structures with typing information are introduced through primitive concepts and primitive roles; on the other hand, views on the objects in the knowledge base are provided through defined concepts.

We propose to account for this conceptual separation by partitioning the TBox into two components for primitive and defined concepts, which we call the *schema* and the *view* part. We envision the two parts to differ with respect to the language for concepts, the statements allowed, and the semantics.

We argue that by this separation we achieve more conceptual clarity about the role of primitive and defined concepts and the semantics of terminological cycles. Moreover, three case studies show the computational benefits to be gained from the refined architecture.

1 Introduction

Research on terminological reasoning usually presupposes the following abstract architecture, which reflects quite well the structure of existing systems. There is a logical representation language that allows for two kinds of statements: in the TBox or *terminology*, concept descriptions are introduced, and in the ABox or *world description*, individuals are characterized in terms of concept membership and role relationship. This abstract architecture has been the basis for the design of systems, the development of algorithms, and the investigation of the computational properties of inferences.

Given this setting, there are three parameters that characterize a terminological system: (*i*) the language for concept descriptions, (*ii*) the form of the statements allowed, and (*iii*) the semantics given to concepts and statements. Research tried to improve systems by modifying these three parameters. But in all existing systems and almost all theoretical studies language and semantics have been kept uniform.[1]

The results of these studies were unsatisfactory in at least two respects. First, it seems that tractable inferences are only possible for languages with little expressivity. Second, no consensus has been reached about the semantics of terminological cycles, although in applications the need to model cyclic dependencies between classes of objects arises constantly.

Based on an ongoing study of applications of terminological systems, we suggest to refine the two-layered architecture consisting of TBox and ABox. Our goal is twofold: on the one hand we want to achieve more conceptual clarity about the role of primitive and defined concepts and the semantics of terminological cycles; on the other hand, we want to improve the tradeoff between expressivity and worst case complexity. Since our changes are not primarily motivated by mathematical considerations but by the way systems are used, we expect to come up with a more practical system design.

In the applications studied we found that the TBox has to meet two functions at a time. One is to declare frame-like structures by introducing primitive concepts and roles together with typing information like isa-relationships between concepts, or range restrictions and number restrictions of roles. *E.g.,* suppose we want to model a company environment. Then we may introduce the concept Employee as a specialization of Person, having exactly one name of type Name and at least one affiliation of type Department. This is similar to class declarations in object-oriented systems. For this purpose, a simple language is sufficient. Cycles occur naturally in modeling tasks, *e.g.*, the boss of an Employee is

*This work was partly supported by the Commission of the European Union under ESPRIT BRA 6810 (Compulog 2), by the German Ministry of Research and Technology under grant ITW 92-01 (TACOS), and by the CNR (Italian Research Council) under Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, LdR "Ibridi."

[1]In [Lenzerini and Schaerf, 1991] a combination of a weak language for ABoxes and a strong language for queries has been investigated.

also an Employee. Such declarations have no definitional import, they just restrict the set of possible interpretations.

The second function of a TBox is to define new concepts in terms of primitive ones by specifying necessary *and* sufficient conditions for concept membership. This can be seen as defining *abstractions* or *views* on the objects in the knowledge base. Defined concepts are important for querying the knowledge base and as left-hand sides of trigger rules. For this purpose we need more expressive languages. If cycles occur in this part they must have definitional import.

As a consequence of our analysis we propose to split the TBox into two components: one for declaring frame structures and one for defining views. By analogy to the structure of databases we call the first component the *schema* and the second the *view* part. We envision the two parts to differ with respect to the language, the form of statements, and the semantics of cycles.

The schema consists of a set of primitive concept introductions, formulated in the *schema language*, and the view part by a set of concept definitions, formulated in the *view language*. In general, the schema language will be less expressive than the view language. Since the role of statements in the schema is to restrict the interpretations we want to admit, first order semantics, which is also called descriptive semantics in this context (see Nebel 1991), is adequate for cycles occurring in the schema. For cycles in the view part, we propose to choose a semantics that defines concepts uniquely, *e.g.*, least or greatest fixpoint semantics.

The purpose of this work is not to present the full-fledged design of a new system but to explore the options that arise from the separation of TBoxes into schema and views. Among the benefits to be gained from this refinement are the following three. First, the new architecture has more parameters for improving systems, since language, form of statements, and semantics can be specified differently for schema and views. So we found a combination of schema and view language with polynomial inference procedures whereas merging the two languages into one would have led to intractability. Second, we believe that one of the obstacles to a consensus about the semantics of terminological cycles has been precisely the fact that no distinction has been made between primitive and defined concepts. Moreover, intractability results for cycles mostly refer to inferences with defined concepts. We proved that reasoning with cycles is easier when only primitive concepts are considered. Third, the refined architecture allows for more differentiated complexity measures, as shown later in the paper.

In the following section we outline our refined architecture for a TKRS, which comprises *three* parts: the *schema*, the *view taxonomy*, and the *world description*, which comprise primitive concepts, defined concepts and assertions in traditional systems. In the third section we show by three case studies that adding a simple schema with cycles to existing systems does not increase the complexity of reasoning.

2 The Refined Architecture

We start this section by a short reminder on concept languages. Then we discuss the form of statements and their semantics in the different components of a TKRS. Finally, we specify the reasoning services provided by each component and introduce different complexity measures for analyzing them.

2.1 Concept Languages

In concept languages, complex concepts (ranged over by C, D) and complex roles (ranged over by Q, R) can be built up from simpler ones using concept and role forming constructs (see Tables 1 and 2 a set of common constructs). The basic syntactic symbols are (i) *concept names*, which are divided into *schema names* (ranged over by A) and *view names* (ranged over by V), (ii) *role names* (ranged over by P), and (iii) *individual names* (ranged over by a, b). An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of the *domain* $\Delta^{\mathcal{I}}$ and the *interpretation function* $\cdot^{\mathcal{I}}$, which maps every concept to a subset of $\Delta^{\mathcal{I}}$, every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and every individual to an element of $\Delta^{\mathcal{I}}$ such that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ for different individuals a, b (*Unique Name Assumption*). Complex concepts and roles are interpreted according to the semantics given in Tables 1 and 2, respectively.

In our architecture, there are two different concept languages in a TKRS, a *schema language* for expressing schema statements and a *view language* for formulating views and queries to the system.

2.2 The Three Components

We first focus our attention to the schema. The schema introduces concept and role names and states elementary type constraints. This can be achieved by *inclusion axioms* having one of the forms:

$$A \sqsubseteq D, \qquad P \sqsubseteq A_1 \times A_2,$$

where A, A_1, A_2 are schema names, P is a role name, and D is a concept of the schema language. Intuitively, the first axiom states that all instances of A are also instances of D. The second axiom states that the role P has domain A_1 and range A_2. A *schema* $\mathcal{S}$ consists of a finite set of schema axioms.

Inclusion axioms impose only necessary conditions for being an instance of the schema name on the left-hand side. For example, the axiom "Employee $\sqsubseteq$ Person" declares that every employee is a person, but does not give a sufficient condition for being an employee.

A schema may contain *cycles* through inclusion axioms (see Nebel 1991 for a formal definition). So one may state that the bosses of an employee are themselves employees, writing "Employee $\sqsubseteq$ $\forall$boss.Employee." In general, existing systems do not allow for terminological cycles, which is a serious restriction, since cycles are ubiquitous in domain models.

There are two questions related to cycles: the first is to fix the semantics and the second, based on this, to come up with a proper inference procedure. As to the semantics, we argue that axioms in the

Construct Name	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
singleton set	$\{a\}$	$\{a^{\mathcal{I}}\}$
intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
universal quantification	$\forall R.C$	$\{d_1 \mid \forall d_2 : (d_1, d_2) \in R^{\mathcal{I}} \rightarrow\ d_2 \in C^{\mathcal{I}}\}$
existential quantification	$\exists R.C$	$\{d_1 \mid \exists d_2 : (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}$
existential agreement	$\exists Q \doteq R$	$\{d_1 \mid \exists d_2.(d_1, d_2) \in Q^{\mathcal{I}} \wedge (d_1, d_2) \in R^{\mathcal{I}}\}$
number restrictions	$(\geq n\, R)$	$\{d_1 \mid \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \geq n\}$
	$(\leq n\, R)$	$\{d_1 \mid \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \leq n\}$

Table 1: Syntax and semantics of concept forming constructs.

Construct Name	Syntax	Semantics
inverse role	P^{-1}	$\{(d_1, d_2) \mid (d_2, d_1) \in P^{\mathcal{I}}\}$
role restriction	$(R{:}\,C)$	$\{(d_1, d_2) \mid (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}$
role chain	$Q \circ R$	$\{(d_1, d_3) \mid \exists d_2.(d_1, d_2) \in Q^{\mathcal{I}} \wedge (d_2, d_3) \in R^{\mathcal{I}}\}$
self	ϵ	$\{(d_1, d_1) \mid d_1 \in \Delta^{\mathcal{I}}\}$

Table 2: Syntax and semantics of role forming constructs.

schema have the role of narrowing down the models we consider possible. Therefore, they should be interpreted under descriptive semantics, *i.e.*, like in first order logic: an interpretation $\mathcal{I}$ satisfies an axiom $A \sqsubseteq D$ if $A^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and it satisfies $P \sqsubseteq A_1 \times A_2$ if $P^{\mathcal{I}} \subseteq A_1^{\mathcal{I}} \times A_2^{\mathcal{I}}$. The interpretation $\mathcal{I}$ is a model of the schema $\mathcal{S}$ if it satisfies all axioms in $\mathcal{S}$. The problem of inferences will be dealt with in the next section.

The *view part* contains *view definitions* of the form

$$V \doteq C,$$

where V is a view name and C is a concept in the view language. Views provide abstractions by defining new classes of objects in terms of the concept and role names introduced in the schema. We refer to "$V \doteq C$" as the *definition* of V. The distinction between schema and view names is crucial for our architecture. It ensures the separation between schema and views.

A *view taxonomy* $\mathcal{V}$ is a finite set of view definitions such that (*i*) for each view name there is at most one definition, and (*ii*) each view name occurring on the right hand side of a definition has a definition in $\mathcal{V}$.

Differently from schema axioms, view definitions give necessary *and* sufficient conditions. As an example of a view, one can describe the bosses of the employee Bill as the instances of "$\mathsf{BillsBosses} \doteq \exists \mathsf{boss\text{-}of}.\{\mathsf{BILL}\}$."

Whether or not to allow cycles in view definitions is a delicate design decision. Differently from the schema, the role of cycles in the view part is to state recursive definitions. For example, if we want to describe the group of individuals that are above Bill in the hierarchy of bosses we can use the definition "$\mathsf{BillsSuperBosses} \doteq \mathsf{BillsBosses} \sqcup \exists \mathsf{boss\text{-}of}.\mathsf{BillsSuperBosses}$." But note that this does not yield a definition if we assume descriptive semantics because for a fixed interpretation of BILL and of the role $\mathsf{boss\text{-}of}$ there may be several ways to interpret $\mathsf{BillsSuperBosses}$ in such a way that the above equality holds. In this example, we only obtain the intended meaning if we assume least fixpoint semantics. This observation holds more generally: if cycles are intended to uniquely define concepts then descriptive semantics is not suitable. However, least or greatest fixpoint semantics or, more generally, a semantics based on the μ-calculus yield unique definitions (see Schild 1994). Unfortunately, algorithms for subsumption of views under such semantics are known only for fragments of the concept language defined in Tables 1 and 2.

In this paper, we only deal with acyclic view taxonomies. In this case, the semantics of view definitions is straightforward. An interpretation $\mathcal{I}$ satisfies the definition $V \doteq C$ if $V^{\mathcal{I}} = C^{\mathcal{I}}$, and it is a model for a view taxonomy $\mathcal{V}$ if $\mathcal{I}$ satisfies all definitions in $\mathcal{V}$.

A state of affairs in the world is described by *assertions* of the form

$$C(a), \qquad R(a, b),$$

where C and R are concept and role descriptions in the view language. Assertions of the form $A(a)$ or $P(a, b)$, where A and P are names in the schema, resemble basic facts in a database. Assertions involving complex concepts are comparable to view updates.

A *world description* $\mathcal{W}$ is a finite set of assertions. The semantics is as usual: an interpretation $\mathcal{I}$ satisfies $C(a)$ if $a^{\mathcal{I}} \in A^{\mathcal{I}}$ and it satisfies $R(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$; it is a model of $\mathcal{W}$ if it satisfies every assertion in $\mathcal{W}$.

Summarizing, a knowledge base is a triple $\Sigma = \langle \mathcal{S}, \mathcal{V}, \mathcal{W} \rangle$, where $\mathcal{S}$ is a schema, $\mathcal{V}$ a view taxonomy, and $\mathcal{W}$ a world description. An interpretation $\mathcal{I}$ is a model of a knowledge base if it is a model of all three components.

2.3 Reasoning Services

For each component, there is a prototypical reasoning service to which the other services can be reduced.

Schema Validation: Given a schema $\mathcal{S}$, check whether there exists a model of $\mathcal{S}$ that interprets every schema name as a nonempty set.

View Subsumption: Given a schema $\mathcal{S}$, a view taxonomy $\mathcal{V}$, and view names V_1 and V_2, check whether $V_1^{\mathcal{I}} \subseteq V_2^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{S}$ and $\mathcal{V}$;

Instance Checking: Given a knowledge base Σ, an individual a, and a view name V, check whether $a^{\mathcal{I}} \in V^{\mathcal{I}}$ holds in every model $\mathcal{I}$ of Σ.

Schema validation supports the knowledge engineer by checking whether the skeleton of his domain model is consistent. Instance checking is the basic operation in querying a knowledge base. View subsumption helps in organizing and optimizing queries (see *e.g.* Buchheit *et al.* 1994). Note that the schema $\mathcal{S}$ has to be taken into account in all three services and that the view taxonomy $\mathcal{V}$ is relevant not only for view subsumption, but also for instance checking. In systems that forbid cycles, one can get rid of $\mathcal{S}$ and $\mathcal{V}$ by expanding definitions. This is not possible when $\mathcal{S}$ and $\mathcal{V}$ are cyclic.

2.4 Complexity Measures

The separation of the core of a TKRS into three components allows us to introduce refined complexity measures for analyzing the difficulty of inferences.

The complexity of a problem is generally measured with respect to the size of the whole input. However, with regard to our setting, three different pieces of input are given, namely the schema, the view taxonomy, and the world description. For this reason, different kinds of complexity measures may be defined, similarly to what has been suggested in [Vardi, 1982] for queries over relational databases. We consider the following measures (where $|X|$ denotes the size of X):

Schema Complexity: the complexity as a function of $|\mathcal{S}|$;

View Complexity: the complexity as a function of $|\mathcal{V}|$;

World Description Complexity: the complexity as a function of $|\mathcal{W}|$;

Combined Complexity: the complexity as a function of $|\mathcal{S}| + |\mathcal{V}| + |\mathcal{W}|$.

Combined complexity takes into account the whole input. The other three instead consider only a part of the input, so they are meaningful only when it is reasonable to suppose that the size of the other parts is negligible. For instance, it is sensible to analyze the schema complexity of view subsumption because usually the schema is much bigger than the two views which are compared. Similarly, one might be interested in the world description complexity of instance checking whenever one can expect $\mathcal{W}$ to be much larger than the schema and the view part.

It is worth noticing that for every problem combined complexity, taking into account the whole input, is at least as high as the other three. For example, if the complexity of a problem is $O(|\mathcal{S}| \cdot |\mathcal{V}| \cdot |\mathcal{W}|)$, its combined complexity is cubic, whereas the other ones are linear. Similarly, if the complexity of a given problem is $O(|\mathcal{S}|^{|\mathcal{V}|})$, both its combined complexity and its view complexity are exponential, its schema complexity is polynomial, and its world description complexity is constant.

In this paper, we use combined complexity to compare the complexity of reasoning in our architecture with the traditional one. Moreover, we use schema complexity to show how the presence of a large schema affects the complexity of the reasoning services previously defined. View and world description complexity have been investigated (under different names) in [Nebel, 1990, Baader, 1990] and [Schaerf, 1993, Donini *et al.*, 1994], respectively.

3 The Case Studies

We studied some illustrative examples that show the advantages of the architecture we propose. We extended three systems by a simple cyclic schema language and analyzed their computational properties.

As argued before, a schema language should at least be expressive enough for declaring subconcept relationships, restricting the range of roles, and specifying roles to be necessary (at least one value) or single valued (at most one value). These requirements are met by the language $\mathcal{SL}$, which was introduced in [Buchheit *et al.*, 1994] and that is defined by the following syntax rule:

$$C, D \longrightarrow A \mid \forall P.A \mid (\geq 1\ P) \mid (\leq 1\ P).$$

Obviously, it is impossible to express in $\mathcal{SL}$ that a concept is empty. Therefore, schema validation in $\mathcal{SL}$ is trivial. Also, subsumption of concept names is polynomially decidable.

We proved that inferences become harder for extensions of $\mathcal{SL}$. If we add inverse roles, schema validation remains trivial, but subsumption of schema names becomes NP-hard. If we add any construct by which one can express the empty concept—like disjointness axioms—schema validation becomes NP-hard. However, in our opinion this does not mean that extensions of $\mathcal{SL}$ are not feasible. For some extensions, there are natural restrictions on the form of schemas that decrease the complexity. Also, it is not clear whether realistic schemas will contain structures that require complex computations.

In all the three cases studied, the schema language is $\mathcal{SL}$. For the view language, we propose three different languages derived from three actual systems described in the literature, namely the deductive object-oriented database system CONCEPTBASE [Jarke, 1992], and the terminological systems KRIS [Baader and Hollunder, 1991] and CLASSIC

[Borgida *et al.*, 1989]. We investigated the computational properties of the reasoning services with respect to $\mathcal{SL}$-schemas. We aimed at showing two results: (*i*) reasoning w.r.t. schema complexity is always tractable, (*ii*) combined complexity is not increased by the presence of terminological cycles in the schema.

In all three cases, we assume that view names are allowed in membership assertions and that the view taxonomy is acyclic. In this setting, every view name can be substituted with its definition. For this reason, from this point on, we suppose that view concepts are completely expanded. Therefore, when evaluating the complexity, we replace the size of the view part by the size of the concept representing the view.

We have found the following results for the three systems in which $\mathcal{SL}$ is the schema language and the concept language the abstraction of the query language of ConceptBase introduced in [Buchheit *et al.*, 1994], or the language offered by kris or classic, respectively.

ConceptBase: instance checking is in PTIME w.r.t. combined complexity (view subsumption has been proved in PTIME in [Buchheit *et al.*, 1994]).

kris: view subsumption and instance checking are PSPACE-complete problems w.r.t. combined complexity and PTIME problems w.r.t. schema complexity.

classic: view subsumption and instance checking are problems in PTIME w.r.t. combined complexity.

We conclude that adding (possibly cyclic) schema information does not change the complexity of reasoning within the systems taken into account.

4 Conclusion

We have proposed to replace the traditional TBox in a terminological system by two components: a schema, where primitive concepts describing frame-like structures are introduced, and a view part that contains defined concepts. We feel that this architecture reflects adequately the way terminological systems are used in most applications.

We also think that this distinction can clarify the discussion about the semantics of cycles. Given the different functionalities of the schema and view part, we propose that cycles in the schema are interpreted with descriptive semantics while for cycles in the view part a definitional semantics should be adopted.

In three case studies we have shown that the revised architecture yields a better tradeoff between expressivity and the complexity of reasoning.

The schema language we have introduced might be sufficient in many cases. Sometimes, however, one might want to impose more integrity constraints on primitive concepts than those which can be expressed in it. We see two solutions to this problem: either enrich the language and have to pay by a more costly reasoning process, or treat such constraints in a passive way by only verifying them for the objects in the knowledge base. The second alternative can be given a logical semantics in terms of epistemic operators (see Donini *et al.* 1992).

References

[Baader and Hollunder, 1991] Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proc. PDK-91*, LNAI, pages 67–86, 1991.

[Baader, 1990] Franz Baader. Terminological cycles in KL-ONE-based knowledge representation languages. In *Proc. AAAI-90*, pages 621–626, 1990.

[Borgida *et al.*, 1989] Alexander Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. CLASSIC: A structural data model for objects. In *Proc. ACM SIGMOD*, pages 59–67, 1989.

[Buchheit *et al.*, 1994] Martin Buchheit, Manfred A. Jeusfeld, Werner Nutt, and Martin Staudt. Subsumption between queries to object-oriented databases. *Information Systems*, 19(1):33–54, 1994.

[Donini *et al.*, 1992] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaerf. Adding epistemic operators to concept languages. In *Proc. KR-92*, pages 342–353, 1992.

[Donini *et al.*, 1994] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Deduction in concept languages: From subsumption to instance checking. *Journal of Logic and Computation*, 4(92–93):1–30, 1994.

[Jarke, 1992] M. Jarke. ConceptBase V3.1 User Manual. Aachener Informatik-Berichte 92-17, RWTH Aachen, 1992.

[Lenzerini and Schaerf, 1991] Maurizio Lenzerini and Andrea Schaerf. Concept languages as query languages. In *Proc. AAAI-91*, pages 471–476, 1991.

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

[Nebel, 1991] Bernhard Nebel. Terminological cycles: Semantics and computational properties. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 331–361. Morgan Kaufmann, Los Altos, 1991.

[Schaerf, 1993] Andrea Schaerf. On the complexity of the instance checking problem in concept languages with existential quantification. *Journal of Intelligent Information Systems*, 2:265–278, 1993.

[Schild, 1994] Klaus Schild. Terminological cycles and the propositional μ-calculus. In *Proc. KR-94*, 1994.

[Vardi, 1982] M. Vardi. The complexity of relational query languages. In *Proc. STOC-82*, pages 137–146, 1982.