Theoretical and Architectural Framework for Contextual Modular Knowledge Bases[*]

Krzysztof Goczyła, Aleksander Waloszek, Wojciech Waloszek, and Teresa Zawadzka

Gdańsk University of Technology, Poland
{kris,alwal,wowal,tegra}@eti.pg.gda.pl

Abstract. The paper presents the approach aimed at building modularized knowledge bases in a systematic, context-aware way. The paper focuses on logical modeling of such knowledge bases, including an underlying SIM metamodel. The architecture of a comprehensive set of tools for knowledge-base systems engineering is presented. The tools enable an engineer to design, create and edit a knowledge base schema according to a novel context approach presented elsewhere by the authors. It is explained how a knowledge base built according to SIM (Structured-Interpretation Model) paradigm is processed by a prototypical reasoner Conglo-S, which is a custom version of widely known Pellet reasoner extended with support for modules of ontologies called tarsets (also introduced elsewhere under the name of conglomerates). The user interface of the system is a plug-in to Protégé ontology editor that is a standard tool for development of Semantic Web ontologies. Possible applications of the presented framework to development of knowledge bases for culture heritage and scientific information dissemination are also discussed.

Keywords: knowledge base, modularization, OWL API, ontology editor, reasoned.

1 Introduction

Knowledge bases grow bigger and bigger. The vast amount of information that has to be stored and managed in an intelligent way motivated intensive research in the field of management of large knowledge bases. One of the hot topics in this field is modularization of knowledge bases. In this chapter we present a method of modularization that we invented and intend to use in the SyNaT project (see the footnote of this page). The method is based on the notion of tarset (previously called "conglomerate") that enables a system to manage a knowledge bases in pieces that themselves are (much smaller) knowledge bases. Tarsets can be seen as analogs to single relations or groups of relations in a relational database, that can be handled

[*] This work has been partially supported by the Polish National Centre for Research and Development (NCBiR) under Grant No. SP/I/1/77065/10 by the strategic scientific research and experimental development program: „Interdisciplinary System for Interactive Scientific and Scientific-Technical Information - SyNaT".

separately (e.g. as views), or as a whole. Upon the structure of tarsets different logical structures of a knowledge base can be built. In this chapter we show how the contextual approach to design and management of a knowledge base can be realized using the tarset-based approach in order to obtain a semantically modularized knowledge base.

This chapter is organized as follows. In the Section 2, that is the main contribution of this paper, we present theoretical background of our approach, with recall of basic notions that lie behind contexts and tarsets. We also introduce the concept of couplers that enable a system to interpret different logical dependencies between tarsets (e.g. contextual dependencies). This sections contains also an ontological metamodel for a tarset-aware knowledge base. In Section 3we present implementation aspects of our framework: an ontology editor especially suited to create and edit contextual ontologies, an important component of our framework that is the Conglo-S module which is a tarset-aware Description Logic reasoner that is able to interpret tarset algebra operators, and then the architecture of the framework that we build to implement the ideas presented in the previous sections. Both: Conglo-S and the editor use widely known open-source tools, extending them with facilities needed to support our approach. In Section 4 we discuss our attitude to application of the framework in the SyNaT project. Section 5 gives a comparison with other approaches to modularization of knowledge bases, which justifies our approach, and presents directions for future work.

2 Basics: Underlying SIM Metamodel

In this section we present our vision of what a modular knowledge base really is and how it should be utilized. Theoretical foundation for this is the theory of tarsets, presented in details in [1], [2], [3]. In the first subsection we recall some basic elements of this theory. In the next subsection we introduce the notion of a modular knowledge base schema. Then we present the SIM (Structured-Interpretation Model) and discuss some problems connected with designing SIM knowledge bases.

We assume familiarity of the reader with basic assumptions of Description Logics. From among less widely used terms we exploit the notion of *vocabulary* (aka. signature), which is a triple ($\boldsymbol{C}$, $\boldsymbol{R}$, $\boldsymbol{I}$) consisting of sets of names of concepts ($\boldsymbol{C}$), roles ($\boldsymbol{R}$), and individuals ($\boldsymbol{I}$) respectively. We assume that every Tarski-style interpretation of a vocabulary assigns each concept name a subset of a domain, each role a set of pairs of domain elements, and each individual name an element of a domain.

2.1 Theory of Tarsets

The theory of tarsets was already presented [4]. Here we recall only some of its properties important for the rest of the paper. A knowledge base is defined as a freely chosen subset of the set of all possible tarsets $\mathbf{K}$. An element of this set, a tarset M, is defined as a pair (S, W) of a vocabulary S and a set W containing all intended models of S. Using $S(M)$ and $W(M)$ we describe the two parts of a tarset M. "Intended models" mean the models of S which are allowed by the creator of a given tarset.

Tarsets are semantic modules and are defined in a way which disregards the exact form of a language (like DL). They focuses only on (Tarski-style) interpretations. However, the only known way to describe them is a language. So we say that a tarset satisfies a particular sentence α, denoted $M \models \alpha$, iff $\forall \mathcal{I} \in W(M)$: $\mathcal{I} \models \alpha$. Now we can regard a tarset M as a representation of an ontology, defined as a set of sentences, whose all of the sentences are satisfied by every interpretation contained in $W(M)$.

The tarsets contained by the set **K** are related to each other. There exists a way to describe these relationships. The theory defines an algebra of tarsets allowing to cover some of them. We can perceive the operations of the algebra as predicates describing relations between tarsets or as functions producing new tarsets from old ones. The list of the operations is as follows:

Intersection:	$M_1 \cap M_2 = (S(M_1) \cup S(M_2), W(M_1) \cap W(M_2))$
Union:	$M_1 \cup M_2 = (S(M_1) \cup S(M_2), W(M_1) \cup W(M_2))$
Difference:	$M_1 - M_2 = (S(M_1) \cup S(M_2), W(M_1) - W(M_2))$
Renaming:	$\rho_\gamma(M) = (\gamma(S(M)), \gamma(W(M)))$
Projection:	$\pi_S(M) = (S, \{\mathcal{I}\vert S: \mathcal{I} \in W(M)\})$
Selection:	$\sigma_\alpha(M) = (S(M), \{\mathcal{I} \in W(M): \mathcal{I} \models \alpha\})$

Intersection, union and difference are obvious. Renaming ρ_γ involves a function γ giving a new name *a'* to an element *a* of $S(M)$ and assigning all interpretations of *a* to *a'* in $W(M)$. Projection π_S changes the vocabulary $S(M)$ into S and cuts the interpretation in $W(M)$ in the way that all relationships between original concepts, roles, and individuals whose names remain in the vocabulary are preserved. Selection σ_α leaves in $W(M)$ only the interpretation satisfying the sentence α.

Using the algebra we can manipulate our knowledge base by producing new modules (or choosing them from **K**) and in this way changing the state of the base.

2.2 Knowledge Base Schema

From the engineer's point of view such a definition of a knowledge base, as presented above, is insufficient. A properly defined engineer's methodology should allow to constrain users in their activities during all stages of a system's life. There are some essential questions a designer should answer during the design stage:

- Which modules are removable, and which are not?
- Which modules are unchangeable?
- What relationships exist between modules?
- Which relationships are removable, and which are not?
- How to create new modules and new relationships?

All above questions are very important but they take into account only the technical aspect of the knowledge base dynamics. Equally important is the semantic aspect: how those activities correspond to the semantics of the modeled problem:

- What does a module to be added represent?
- What is the meaning of a relationship between two modules?
- What do our actions mean (from semantic point of view) when we add or remove a module/relationship?
- Why some of the modules/relationships are removable and some are not?
- Why some of the modules should be unchangeable?

To address these problems we decided to define a knowledge base schema and a knowledge base instance. Generally speaking, a knowledge base instance is its state. Every time a user changes something in a knowledge base he/she produces its new instance. A knowledge base schema is a set of constraints imposing restrictions on users' activities. From another point of view a schema is a description which categorizes modules and relationships between modules. By creating this description a designer explains semantics of modules and relationships.

2.2.1 Expressions, Equalities and Inequalities

In order to make clear what a schema is, it is necessary to introduce some new notions. The first is a tarset algebra expression. The definition bases on the fact that the operations of the algebra produce new tarsets from old ones.

Definition 1 (*tarset algebra expression*)
An *expression of the tarset algebra* for the domain D is a formula of the form: $F, G := F \cup G \mid F \cap G \mid F - G \mid \pi_S(F) \mid \rho_\gamma(F) \mid \sigma_\alpha(F) \mid d$, where S, γ, α and d respectively mean a vocabulary, a renaming function, a sentence and an element of D. ■

The simplest choice of D is a subset of **K**. It is the only situation where we can perceive an expression as a complex operation producing a tarset as a result.

The other domains are also very useful, e.g. a set of tarset variables. A tarset variable can be a name chosen from a set of allowed variable names (the set is defined by a specific implementation), which has to be assigned a value from **K** during the instantiation process. The symbol $|v|$ means a value assigned to the variable v.

In order to define equality and inequality of tarsets, we assume that every tarset is reduced. A *reduced tarset* is a tarset whose vocabulary and set of models are also reduced. A *reduced vocabulary* $[S(M)]$ is a vocabulary $S(M)$ modified by removing all words with unrestricted meaning accordingly to all interpretations from $W(M)$ (i.e. such words x for which $M = \pi_{S(M)}(\pi_{S(M) - \{x\}}(M))$). Then a *reduced set of models* $[W(M)]$ is produced from $W(M)$ by projection to the reduced vocabulary.

Such an assumption facilitates comparing tarsets because it allows for neglecting insignificant differences between vocabularies. A greater tarset is simply a tarset with greater set of models (i.e. $M_1 > M_2 \Leftrightarrow W(M_1) \supseteq W(M_2)$). Moreover, with this assumption, it is easy to define an *empty tarset*, which is the only one in the space of reduced tarsets and has the form $M_0 = \{\varnothing, \varnothing\}$.

Now we can define *predicative expressions*: tarset equality and inequality:

Definition 2 (*tarset equality and inequality*)
An *algebraic equality* of tarsets for a domain D (a tarset equality) is a formula of the form: $F = G$, and an *algebraic inequality* of tarsets is a formula of the form $F > G$, $F < G$, $F \geq G$ or $F \leq G$, where F, G are expressions of the tarset algebra for the domain D. ■

In general a choice of a domain D is arbitrary as long as its elements refer to tarsets (the domain may, apart from trasets, embrace e.g. tarset functions, or tarset variables). In the most straightforward situation, when $D \subseteq \mathbf{K}$, the satisfaction of a predicative expression depends only on the relationship between tarsets being the results of F and G. The way of satisfying expressions for other domains should be independently defined.

2.2.2 Tarset Knowledge Base, Schema and Instance

It is clear that the old definition of knowledge base is not sufficient to define rules allowing to control the contents of an instance. In order to achieve this goal the definition of a knowledge base should be enriched with new elements. In this subsection we redefine the notion of a knowledge base, taking into consideration its static (a schema) and dynamic (an instance) aspects. The form of the definition is strongly determined by the kinds of possible changes we can distinguish:

1. Changes of the contents of a tarset:
   (a) vocabulary changes, i.e. adding new words (general or individual names); in order to keep monotonicity we do not expect removing words,
   (b) semantic changes, i.e. adding new axioms describing relations between general names from the vocabulary,
   (c) factual changes, i.e. accepting new facts about individual names from the vocabulary.
2. Adding and/or removing tarsets.
3. Adding or removing relationships between tarsets.

These forms of changes will be discussed in next subsections.

Changes of the contents of a tarset
This kind of change is equivalent to addition of a new axiom to a tarset. It is not an addition itself, as a tarset is not a syntactic entity and does not contain sentences. Thus when we speak about "adding a sentence" α, we mean something like executing the operation of selection σ_α. As a result we get a new set of interpretations, that is a new tarset from the set $\mathbf{K}$. According to the old definition the result of such a change is a new knowledge base: a new set of tarsets. The conclusion is that it is much better to perceive a knowledge base as a set of tarset variables than a set of tarsets. The described change would not alter this set but only an assignment of (at least one of) the variables.

The decision that a knowledge base is a set of tarset variables allows us to define a new kind of elements controlling the contents of tarsets.

Definition 3 (*coupler*)
Let V be a set of tarset variables. A *coupler over* V is a predicative expression (tarset equality or inequality) for the domain V. ■

A coupler s is satisfied iff the predicative expression is satisfied after mapping all tarset variables into elements of V. We assume that in a knowledge base the only accepted changes are the changes satisfying all its couplers. The couplers represent the aforementioned relationships between tarsets.

Adding or removing tarsets or relationships between tarsets
This kind of changes is much more difficult to control but is the most essential from the modular knowledge bases point of view. Adding or removing a tarset is connected with the change of the set of tarset variables. This change, on the other hand, imposes a change of the set of couplers. But the last change has also be under control. Thus there is a need to introduce new elements: types of tarsets and types of couplers. The purpose of these elements is to categorize groups of tarsets and couplers w.r.t their properties and to take control over this kind of changes.

Types of tarsets make a structure ordered by the inheritance relation. This structure is called a *hierarchy of tarset types*:

Definition 4 (*hierarchy of tarset types*)
A *hierarchy of tarset types* is a strict partial order (T_K, $\lhd$), where T_K is called a *set of tarset variable types*, and $\lhd$ is called the *inheritance relation*. We say that t_1 *extends* (*inherits from*) t_2 iff $t_1 \lhd t_2$ ($t_1, t_2 \in T_K$). ■

It is possible to put a hierarchy of tarset types on a given set of tarset variables.

Definition 5 (*establishing types*)
Let V_K be a finite set of tarset variables. *Establishing types by putting a hierarchy* (T_K, $\lhd$) *on the set* V_K consists in setting a function f_{in}: $T_K \longrightarrow \mathcal{P}(V_K)$, where $\mathcal{P}(V_K)$ is the powerset of V_K. The function f_{in} takes into account transitivity of the relation $\lhd$ in the way that $\forall t_1, t_2 \in T_K\ \forall v \in V_K\ ((t_1 \lhd t_2 \wedge v \in f_{in}(t_2)) \rightarrow v \in f_{in}(t_1)))$. Every variable v assigned to $t \in T_K$, i.e. such that $v \in f_{in}(t)$, is called a *variable of the type t*. f_{in} is called a *result of establishing types*. ■

It is worth to note that tarset types are types of tarset variables. Two variables of different types may be assigned the same value.

The other kind of types, coupler types, are used to categorize couplers.

Definition 6 (*coupler types*)
Let V_K be a finite set of tarset variables with types established by a structure (T_K, $\lhd$, f_{in}). A *coupler type over* V_K is a predicative expression (tarset equality or inequality) for a domain of slots L. A *slot* $l \in L$ is a pair (v, t), where $t \in T_K$ and $v \in V_K$ or v is a variable, whose range of assignment is $f_{in}(t)$. ■

A given coupler s is of the type t_s iff it is a formula resulting of the substitution of all slots (v, t) in t_s by v, where $v \in V_K$, or by $|v|$ otherwise.

Having coupler types defined, we can introduce the notion of tarset types. A tarset type consists of constraints describing the requirements against tarsets conforming to the type. We can formulate the definition of tarset type constraints as follows:

Definition 7 (*tarset type constraints*)
Let T_S be a set of coupler types. *Tarset type constraints* over T_S are constraints of the form:

cardinality constraint for a given tarset type t is a triple (t_s, l, N), where $t_s \in T_S$, l is a chosen slot, and N is a subset of the set of the natural numbers;

editing constraint for a given tarset type t is a value from the set {*noTerminology*, *noFacts*, *noInstances*}. ■

Now we are ready to define a schema and an instance of a knowledge base. A schema is its fixed part; we assume that by changing a schema we define a new knowledge base.

Definition 8 (*tarset knowledge base schema*)
A *tarset knowledge base schema* is a structure $(K_S, V_{KS}, (T_K, \lhd), f_{inS}, T_S, C, f_o, w_{KS})$, where:

- K_S — subset of $\mathbf{K}$,
- V_{KS} — a set of tarset variables,
- $(T_K, \lhd)$ — a hierarchy of tarset types,
- f_{inS} — the result of establishing $(T_K, \lhd)$ on V_{KS},
- T_S — a set of coupler types,
- C — a set of tarset type constraints over T_S,
- f_C — a function mapping a subset of C to every type from T_K; f_c takes into account transitivity of the relation $\lhd$, mapping to every type all constraints assigned to the inherited types in such a way that $t_1 \lhd t_2 \Rightarrow f_C(t_1) \subseteq f_o(t_2)$,
- w_{KS} — a partial function $V_{KS} \longrightarrow K_S$.

All the sets mentioned above are finite. ■

The elements K_S and V_{KS} allow to predefine a part of an instance during the designing stage. In other words, every schema can contain predefined tarset variables and tarsets (constants). As members of a schema they cannot be removed.

Unlike a schema, which is a fixed part of a base, an instance is its temporary state. Every change generates a new state and a new instance comes into existence (or is chosen from the universe of all possible instances).

Definition 9 (*tarset knowledge base instance*)
An *instance of a tarset knowledge base* $\Sigma = (K_S, V_{KS}, (T_K, \lhd), f_{inS}, T_S, C, f_C, w_{KS})$ is a structure $(\Sigma, (K, V_K, S, w_K), f_{in}, f_{sin})$, where:

- K — subset of $\mathbf{K}$,
- V_K — a set of tarset variables,
- S — a set of couplers,

- w_K — a function $V_K \longrightarrow K$,
- f_{in} — the result of establishing $(T_K, \lhd)$ on V_K,
- f_{sin} — a function $S \longrightarrow T_S$.

All the elements of the structure have to satisfy the following conditions, called the *schema adjustment conditions*:

- *the condition of constants preservation*: $K \supseteq K_S$,
- *the condition of variables preservation*: $V_K \supseteq V_{KS}$,
- *the condition of constant values preservation*: $w_K \supseteq w_{KS}$,
- *the condition of proper tarsets typing*: $f_{in} \supseteq f_{inS}$,
- *the condition of couplers satisfaction*: all the couplers from the set S have to be satisfied,
- *the condition of proper couplers typing*: for every $s \in S$: s has to be of the type f_{sin}(s),
- the condition of constraints preservation: for every tarset type t and every tarset variable v from $f_{in}(t)$, all the constraints from the set $f_C(t)$ have to be satisfied, i.e.:
  - every cardinality constraint (t_s, l, N) is satisfied, when the number of couplers from S, created by substitution of every l by v and such that $f_{sin}(s) = t_s$, belongs to N,
  - every editing constraint E is satisfied. ■

Strict mathematical description of the meaning of an editing constraint E would take too much space. Simplifying, we can say that if E = *noInstances* then it is forbidden to create new tarsets of a given type (i.e. $f_{in}(t) = f_{inS}(t)$); if E = *noTerminology* then adding any terminological axioms to a tarset of a given type is forbidden; if E = *noFacts* then adding new facts is forbidden.

2.3 Ontological Description of the Metamodel

The mathematical metamodel of a tarset knowledge base presented above has been described in the ontological way The ontology is expressed in the OWL DL language and was edited using Protégé (http://protege.stanford.edu/). It consists of three modules: *Algebraization*, *SchemaBasicTBox* and *InstanceBasicTBox*. The modules are standard, flat ontologies connected with commonly used option called *import*.

The module *Algebraization* contains definitions of the elements representing the algebraic and predicative expressions. There are concepts allowing to describe main entities, like *SKBPredicativeExpression* or *SKBExpression*, the operators (e.g. *SKBIntersection*, *SKBUnion*, *SKBProjection*, etc.), operands (e.g. *SKBExpression Variable*) and some auxiliary entities (among others: *AxiomSet*, *NameSet* and operations on them). The relations between concepts are described by 12 roles, e.g.: *hasSKBOperand* (domain: *SKBOperation*, range: *SKBExpression*), *isRestrictedBy* (domain: *SKBSelection*, range: *AxiomSet*), etc. This module is very important as it is the essential part of coupler type definitions.

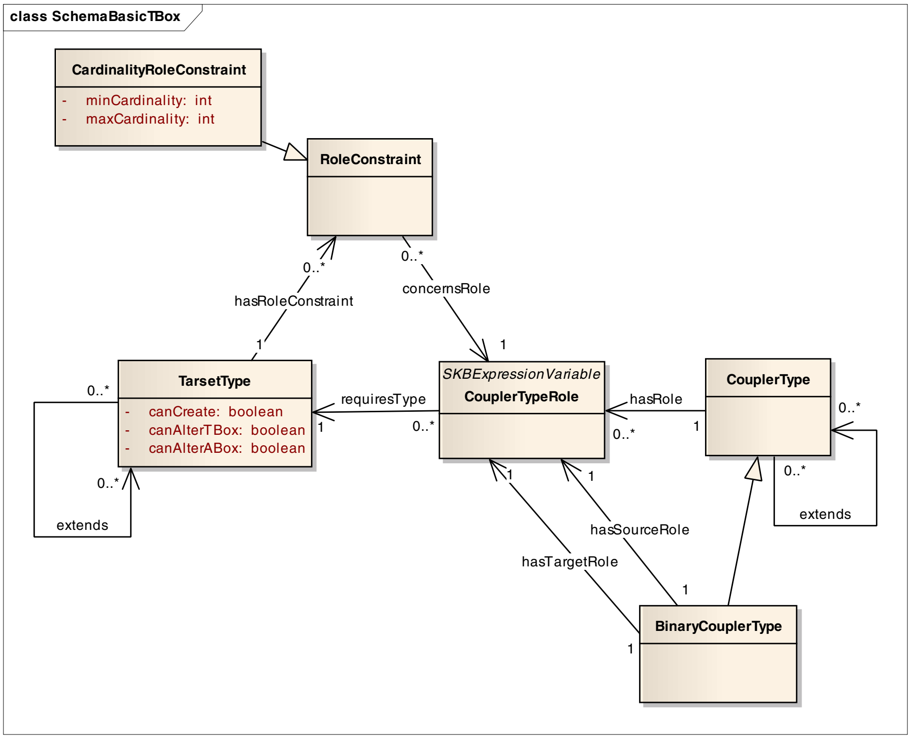


Fig. 1. Metamodel of a knowledge base schema

The module *SchemaBasicTBox* defines the metamodel of a tarset knowledge base schema. It is shown in Fig. 1. Classes represent concepts and associations represent roles. The main concepts are *TarsetType* and *CouplerType*. The former is extended by *BinaryCouplerType*, because such a type is expected to be used most frequently. As we see, the concept *CouplerTypeRole* inherits from the concept *SKBExpressionVariable* and, in this way, implements also the notion of a slot. There is one more role not depicted in this diagram: it is called *isRealizedBy* and connects the concept *CouplerType* with *SKBPredicativeExpression*. This connection is essential for the definition of a coupler type.

The module *InstanceBasicModule* is focused on the notions related to knowledge base instances. It imports *SchemaBasicTBox* as every element of an instance has to be connected with the corresponding element of a schema. The concepts *Tarset* and *Coupler* are connected with *ConceptType* and *CouplerType* respectively with the role *hasType*. The role *assigns* connects *RoleAssignment* with *CouplerTypeRole*. Figure 2 presents the metamodel of a knowledge base instance. From this figure it is easy to see what is the reason for defining *BinaryCouplerType*. The concept *BinaryCoupler* is directly connected with the concept *Tarset* without mediation of the concept *RoleAssignment*. Thanks to this, the schema of an instance is much simpler.

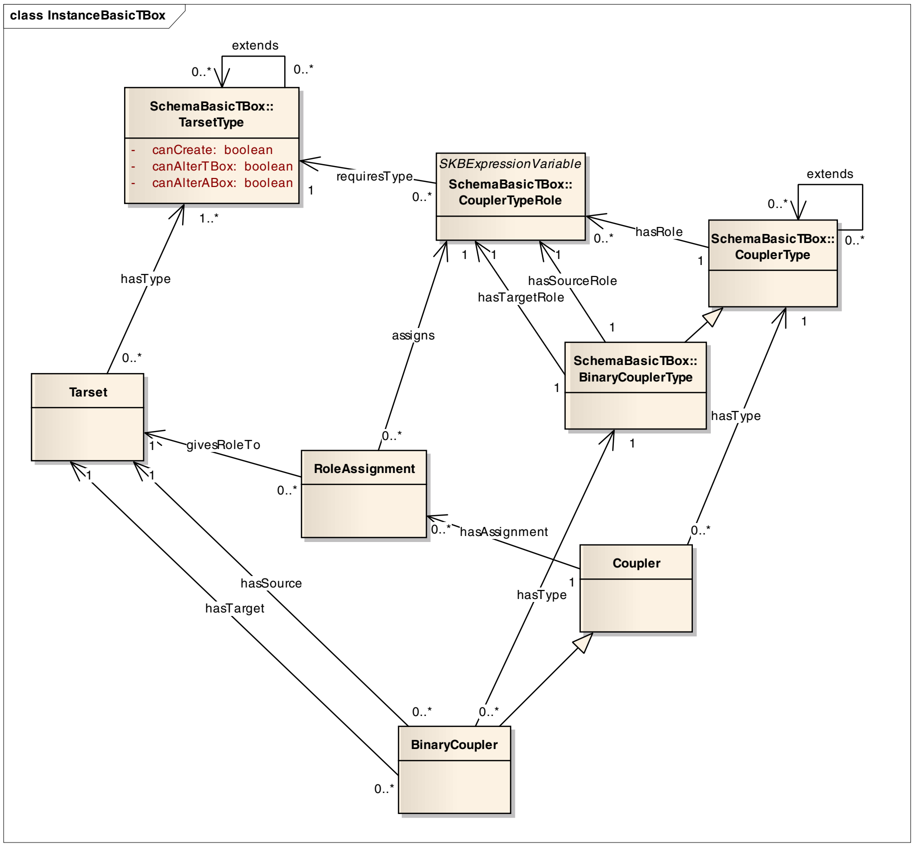


Fig. 2. Metamodel of a knowledge base instance

The main idea behind using the ontological form of a description is to deliver a basis for practical solutions for future implementations. We assume that every implementation of tarset knowledge base will also be described ontologically as an extension of the metamodel. Figure 3 shows a possible architecture.

The block named “Basic terminology” is the metaontology itself. The ontologies inside form the common part of all knowledge bases and are available as an internet resource. Every knowledge base (the blocks “Knowledge Base 1”, “Knowledge Base 2”, etc.) has its own schema (the fixed part) and instance (the dynamic part), both described ontologically. The schema imports the basic terminology. It may be extended with new general terms, but by doing this one cannot change the semantics of the metaontology. Such an extension is called “conservative extension” (see [5]). Any extension of the terminology here has only a local meaning because it is assumed that the tarset reasoning process is aware only of the language defined by the basic terminology.

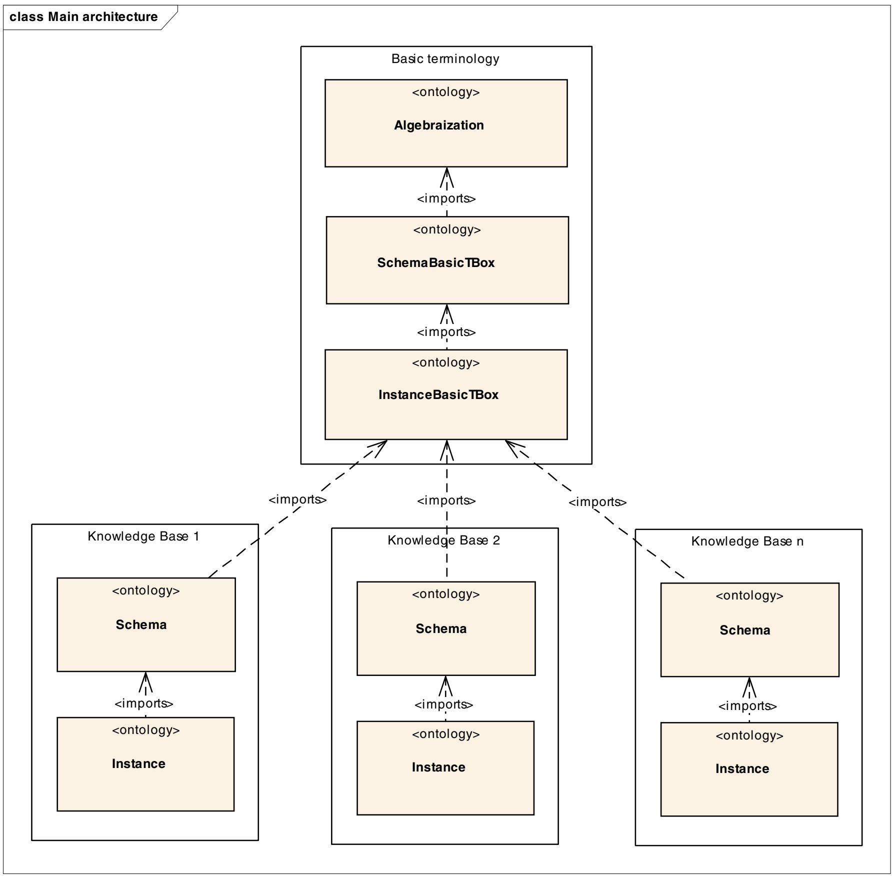


Fig. 3. Architecture of ontological descriptions of tarset knowledge bases

The instance ontology consists of facts (is an ABox), constantly changes (yet preserving monotonicity), and imports the schema ontology. Every tarset and every coupler is represented as an individual, but their contents are kept in separate ontologies, pointed by special annotations. It is convenient to describe the contents of a tarset in the form of sentences using the notion of $\mathcal{L}$-(2)-**D**-representation (see the next part of the paper). This kind of representation assumes that every module is represented by a set of sets of sentences. It is convenient to keep every set of sentences in a separate ontology.

2.4 SIM Contextualized Knowledge Bases

In [4] we described shortly the SIM methodology so now we discuss only the most important of its properties needed to understand the rest of the paper. More information may be found in [6][7][8].

SIM (Structured-Interpretation Model) is an extension of Description Logic formalism allowing to define modular ontologies of a specific form. As a normal DL ontology it consists of a terminology (a contextualized TBox) and a world description (a contextualized ABox). Both these parts are divided into modules which are connected to each other by three types of relations.

A *contextualized TBox* is a poset $(\{T_i\}_{i \in I}, \trianglelefteq)$ with a single least element T_m. The indexed set $\{T_i\}$ contains terminological modules (TBoxes), called context types. Context types are ordered by the inheritance relation $\trianglelefteq$. The T_m is called the *root context type* and is the ancestor of all other context types in the structure.

The semantics of such a structure impose that every successor inherits all axioms defined in its ancestors. It is equivalent to importing the contents of the "less" module to the "greater" module.

A *contextualized ABox of the contextualized TBox* $(\{T_i\}_{i \in I}, \trianglelefteq)$ is a structure that contains three elements: $(\{A_j\}_{j \in J}, f_{inst}, \preccurlyeq)$. The set $\{A_j\}$ contains modules with descriptions of facts (ABoxes). These modules are called context instances. f_{inst} is a function $\{A_j\} \longrightarrow \{T_i\}$, called *instantiation function*, assigning every context instance to a single context type. This connection also resembles an OWL import, since a context instance imports all the contents from its context type and its ancestors.

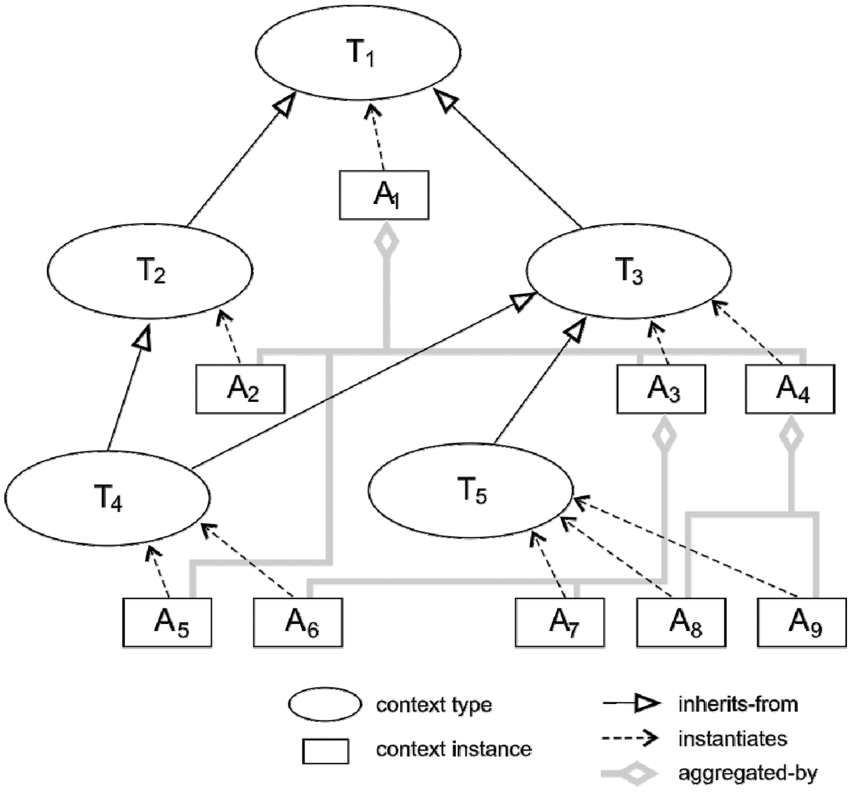


Fig. 4. An example of a SIM knowledge base

The last element of the structure is the aggregation relation $\lessdot$. This relation is also a partial order with exactly one least element.

The both relations ($\trianglelefteq$ and $\lessdot$) and the function f_{inst} determine the paths of flow of conclusions during reasoning. In the case of aggregation relation the flow has the direction from aggregated to aggregating instances. All sentences must be taken into account, but due to the fact that the attached TBox is more general some information must be reinterpreted in more general terms. Figure 4 depicts three aggregating context instances: A_1, A_3 and A_4.

It is worth stressing that if levels of generality (introduced by context types) are properly chosen, we can aggregate information from context instances holding contradictory assertions without making the knowledge base inconsistent.

The main principle of SIM structures is that the higher level of hierarchy, the bigger set of individuals is described in more general way.

Properties of the SIM method allow us to achieve one more aim. It is easy to establish the boundary between a schema (a fixed part of a knowledge base) and an instance (a changeable part). In [8] this problem is discussed more precisely. Here, not going into details, we can say that a *SIM knowledge base schema* is a contextual TBox with so called *admissible places of aggregation* pointing out places where instances of a given context type should be aggregated. On the other hand a *SIM knowledge base instance* is a contextual ABox built regarding to the requirements imposed by the schema.

2.5 SIM Modules as Tarsets

To explain the correlation between the two described above models we have to refer to the analogy with relational databases. While designing a new database, we first prepare an E-R diagram, in which we introduce relationships between entity sets. Each kind of relationships may be expressed in relational algebra, but they are only a fraction of all kind of relationships expressible in this way. In other words, during the high level design we discard variety of possibilities offered by the algebra and focus on selected, most important, kinds of relationships.

We perceive the role of a SIM schema similarly as this of E-R. The model captures the most important factors connected with contextual modeling (like e.g. change of perspective) and, as such, is an ideal candidate for providing a core set of relationships between semantic modules.

It is very important that all the constraints imposed by SIM models are expressible in tarset algebra. This allowed us to create a uniform, general system of defining knowledge base schemas, in which we treat SIM elements as a vital part of every tarset knowledge base (depicted in Fig. 3 as Knowledge Base 1..*n*).

In practice the task has been carried out by development of an additional OWL ontology named *SIMMetaschema*. This ontology has been integrated within the described framework. For each kind of SIM relation (inheritance, instantiation, aggregation) it introduces an appropriate binary coupler type. The tarsets themselves are divided into two types: context types, and context instances.

Figure 5 presents an excerpt from the ontology, where the individual *inheritance* is defined as a type of coupler. Indeed, inheritance is a binary coupler requiring tarsets of the type "context type" for its both roles (ancestor and descendant). The coupler is described by an inequality with two slots (for the two roles) and the inequality itself is a formula $D \leqslant D \cap A$, which simply means that all the axioms from an ancestor are "imported" by a descendant. All the other required inequalities are defined in a similar fashion.

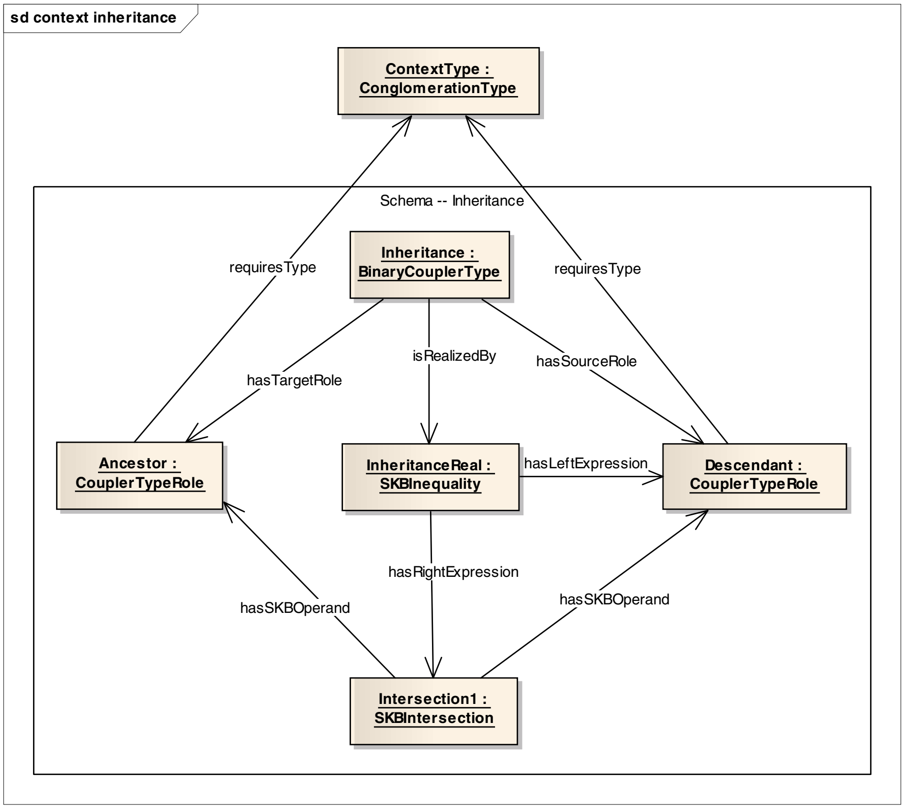

sd context inheritance
ContextType : ConglomerationType
Schema -- Inheritance
Inheritance : BinaryCouplerType
requiresType
requiresType
isRealizedBy
hasTargetRole
hasSourceRole
Ancestor : CouplerTypeRole
InheritanceReal : SKBInequality
hasLeftExpression
Descendant : CouplerTypeRole
hasRightExpression
hasSKBOperand
hasSKBOperand
Intersection1 : SKBIntersection

Fig. 5. An excerpt from the *SIMMetaschema* ontology

The achieved result is notable, as on the one hand it indicates that the tarset algebra is powerful enough to fully represent the SIM model, and on the other hand SIM introduces important relations that are very useful for organizing the design of knowledge bases of various kinds in a systematic way.

The resulting framework integrates both methods and allows for exploiting their strengths. We assume that a typical way of use of the framework is to design a "backbone" of a knowledge base only with use of SIM relations, and then augmenting it in the next steps with more specialized couplers. Such course of actions would be similar to creating an E-R backbone design of a database, and then preparing specialized triggers in order to fulfill more sophisticated requirements.

This framework constitutes a base for a set of tools addressed to designing modular knowledge bases described in the next section of this chapter.

3 The Tools

For the theoretical framework presented above to be realized in software practitioners' environments, a set of supporting tools is needed. This section is devoted to presentation of such a set of tools. We stress that some crucial components of this set are required to fulfill needs of knowledge-based systems engineers; these are: a context-aware modular ontology editor (Section 3.1), an appropriate reasoner (Section 3.2), and a general architecture of the framework with use-case analysis (Section 3.3).

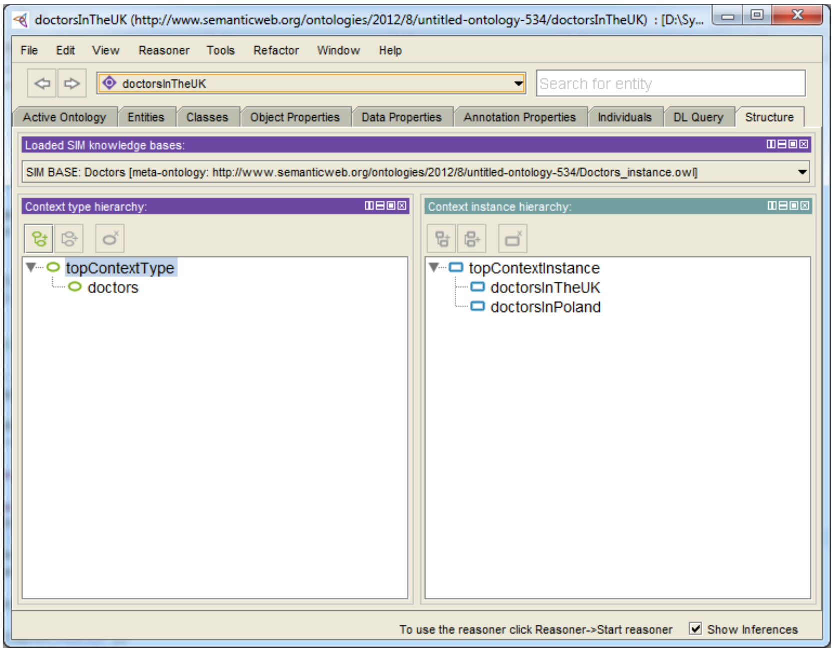
3.1 Protégé Plugin

During the SyNaT project we strive to achieve the aim of developing a suite of tools allowing for designing and using SIM knowledge bases. For the first task we developed an extension (in the form of a set of plugins) in for a well-known and widely used ontology editor – Protégé.

While designing an editor interface of contextual plug-ins, we aimed at obtaining an interface as similar to standard Protégé editor interface as possible. Thus, while presenting this interface, names of consequent elements of interface are taken from standard OWL editor. The terms "window" and "tab" are defined analogically as in operating system. Additionally, we use the term "view" as one of the elements of the specified tab.

The standard set of tabs, normally used in the OWL editor, is supplemented by a new tab (Structure Tab) containing elements allowing to navigate and edit the main structures of a SIM knowledge base. The functionality of the standard set is unchanged: it allows to view and edit OWL ontologies. As SIM modules are, in general, ontologies, the set is also useful for editing their contents. In some cases, though, some possibilities are restricted (e.g. inserting assertions into TBox modules has been prohibited).

The Structure tab help the user with creating the ontological description of the knowledge base (cf. Sec 2.3) by hiding its details and presenting the structure of a SIM knowledge base. The view named Context Type Hierarchy shows the hierarchy of context types in the similar way to the Class Hierarchy view showing the hierarchy of concepts. Selecting any context type causes the hierarchy of context instances being instances of selected context type to be displayed in the view named Context Instances. Additional view allows for designing several knowledge bases within one instance of Protégé editor, and choosing one as an active base.


doctorsInTheUK (http://www.semanticweb.org/ontologies/2012/8/untitled-ontology-534/doctorsInTheUK) : [D:\Sy...

Loaded SIM knowledge bases:

SIM BASE: Doctors [meta-ontology: http://www.semanticweb.org/ontologies/2012/8/untitled-ontology-534/Doctors_instance.owl]

To use the reasoner click Reasoner->Start reasoner ☑ Show Inferences

Fig. 6. Structure tab selected within Protégé editor

3.2 Reasoning with Conglo-S

Conglo-S inference engine (its name being a shortcut for Conglomerated Sentencer) is a reasoner integrated with Protégé system. It is designed for processing tarset knowledge bases with use of one standard OWL reasoning services (currently, by default, it uses Pellet [12] as its internal reasoner, though it is possible to configure it—with use of Preferences Tab—to use any of reasoners integrated with Protégé). Conglo-S reasoner is an extension of a previously developed tool, and the extension is towards handling schemas: i.e. module variables and couplers.

3.2.1 Sentential Representation of Tarsets

In order to meet its basic requirements, Conglo-S needs to convert the semantic modules to a form which will be readable by standard reasoning systems. For this purpose we exploit the theory of *sentential representation* of tarsets.

It is obvious that there are situations when a single module may be represented with a set of sentences expressed in some language $\mathcal{L}$. For such representation one must be able to create exactly the set of satisfying interpretations as for the original module. The notion of $\mathcal{L}$-representability formalizes the idea (by $Sig(O)$, where O is a set of sentences, we understand a vocabulary of all terms used in the sentences):

Definition 10 (*$\mathcal{L}$-representability* and *$\mathcal{L}$-representation*)

A set of sentences $O \subseteq \mathcal{L}$ is an *$\mathcal{L}$-representation* of a tarset M iff $Sig(O) \subseteq \mathbf{S}(M)$ and $\mathbf{W}(M) = \{\mathcal{I}\colon \mathcal{I} \vDash O\}$.

Tarset M is *$\mathcal{L}$-representable* iff there exists a set of sentences $O \subseteq \mathcal{L}$ being its *$\mathcal{L}$-representation.* ■

While some operations (like $\cap$) preserve the $\mathcal{L}$-representability (i.e. while performed on $\mathcal{L}$-representable modules, they produce a module which is also $\mathcal{L}$-representable), it is not always the case, as it can be seen in the following example.

Consider a module $M = M(\{A \sqsubseteq B\}) \cup M(\{B \sqsubseteq A\})$. While it seems that it may be represented with an empty set of sentences (because from M we cannot draw any sensible conclusion which can be expressed with a single sentence), in fact it cannot, because M carries some subtle information, which can be seen by performing $M \cap M(\{A \sqcap \neg B(a), \neg A \sqcap B(b)\})$: in the result, we obtain the empty module.

The second source of trouble with representing tarsets in sentential form is connected with the operation of projecting. It may turn out that the reduced signature of the module is "not enough" to represent all the interrelationships between remaining terms. A simple example of this is the module $M = \pi_{\{A,\,B\}}(M(\{A \sqsubseteq B, A(a)\}))$. The original module contained an assertion $A(a)$ whose effect on the set of models was that it excluded all the interpretations with empty A (and, consequently, B). There is no way of expressing such a constraint while using only the names A and B.

The above discussion indicates that the notion of $\mathcal{L}$-representability has to be extended at least towards handling alternative sets of models (like in the case of the union) and towards handling projection. The first problem can be solved by using multiple sets of sentences instead of a single one. Each set can now determine a separate set of models, and can be treated as a one of several "alternatives". The second problem is tackled with by introducing a set of "auxiliary" names from a special set **D** which can be used regardless of a signature of a module. The simultaneous use of both proposed solutions leads to the new notion of $\mathcal{L}$-(2)-**D**-representability.

Definition 11 ($\mathcal{L}$-(2)-D-*representabiliy*)

Tarset M is *$\mathcal{L}$-(2)-**D**-representable* iff there exists a (finite) set of sets of sentences $\mathcal{S} = \{O_i\}_{i \in [1..k]}$, $k \in N$, each $O_i \subseteq \mathcal{L}$, being its *$\mathcal{L}$-(2)-**D**-representation* (denoted $M \sim_{\mathcal{L}\text{-(2)-}\mathbf{D}} \mathcal{S}$), which means that $\bigcup_{i \in [1..k]} Sig(O_i) \subseteq \mathbf{S}(M) \cup \mathbf{D}$ and $\mathbf{W}(M) = (\bigcup_{i \in [1..k]} \{\mathcal{I}\colon \mathcal{I} \vDash O_i\})|\mathbf{S}(M)$. ■

It can be shown that with such a definition of $\mathcal{L}$-(2)-**D**-representability the core of algebraic operations preserves this feature. An outline of the proof is presented below:

1. If $M_1 \sim_{\mathcal{L}\text{-(2)-}\mathbf{D}} \mathcal{S}_1$, $\mathcal{S}_1 = \{O_{1:i}\}_{i \in [1..k]}$ and $M_2 \sim_{\mathcal{L}\text{-(2)-}\mathbf{D}} \mathcal{S}_2$, $\mathcal{S}_2 = \{O_{2:j}\}_{j \in [1..m]}$, then $M_1 \cup M_2 \sim_{\mathcal{L}\text{-(2)-}\mathbf{D}} \mathcal{S}_1 \cup \mathcal{S}_2$.
2. If $M_1 \sim_{\mathcal{L}\text{-(2)-}\mathbf{D}} \mathcal{S}_1$, $\mathcal{S}_1 = \{O_{1:i}\}_{i \in [1..k]}$ and $M_2 \sim_{\mathcal{L}\text{-(2)-}\mathbf{D}} \mathcal{S}_2$, $\mathcal{S}_2 = \{O_{2:j}\}_{j \in [1..m]}$, then $M_1 \cap M_2 \sim_{\mathcal{L}\text{-(2)-}\mathbf{D}} \{O_{1:i} \cup O_{2:j}\colon i \in [1..k], j \in [1..m]\}$.
3. If $M \sim_{\mathcal{L}\text{-(2)-}\mathbf{D}} \mathcal{S}$, $\mathcal{S} = \{O_i\}_{i \in [1..k]}$, then $\pi_{\mathbf{S}}(M) \sim_{\mathcal{L}\text{-(2)-}\mathbf{D}} \{\gamma_{\mathbf{S}(M) - \mathbf{S} \to \mathbf{D}}(O_i)\colon i \in [1..k]\}$; where by $\gamma_{\mathbf{S}(M) - \mathbf{S} \to \mathbf{D}}$ we understand a function renaming terms not included in **S** to terms from **D**.
4. If $M \sim_{\mathcal{L}\text{-(2)-}\mathbf{D}} \mathcal{S}$, $\mathcal{S} = \{O_i\}_{i \in [1..k]}$, then $\rho_\gamma(M) \sim_{\mathcal{L}\text{-(2)-}\mathbf{D}} \{\gamma(O_i)\colon i \in [1..k]\}$.

Various reasoning tasks can be performed with use of $\mathcal{L}$-(2)-**D**-representation. For instance checking the consistency of the tarset can be performed by checking the consistency of each of the sets in the representation. Consequently, to check if the module entails a given axiom α, one has to check if every set of the representation entails α.

3.2.2 Module Variables and Couplers in Conglo-S

Conglo-S is extended by functions to manage a knowledge base schema. Apart from administering “contents” of each module, which is done in a way sketched in the previous section, Conglo-S is able to handle schemas: module variables with contents changing over time, and couplers, which bind together specific variables with a relationship described by an algebraic inequality.

A user of the system is presented with an interface which allows for creating, updating, and dropping modules. Originally each module is perceived by the user as an ontology (or a set of sentences). Update operations comprise of adding new sentences and retracting the previously added ones. The sentences provided by a user forms the “core” part of each module variable in the knowledge base.

The schema of the knowledge base may be altered by addition (or by dropping) a coupler. Conglo-S currently allows only for using couplers in the form of $M < E$, where M is a single module variable, and E is an algebraic expression which may contain all the defined variables.

Conglo-S interprets couplers in a fix-point fashion. It means that, after each update the contents of every module variable is recalculated by “firing” every coupler (by performing the assignment $M := E$), and the operation is repeated until the fix-point is reached. While the “core” module contents is preserved, for the sake of enabling updates, the contents visible to the user is in fact the result of performing algebraic operations contained in the couplers.

3.3 Architectural Considerations

The tools presented in the previous subsections were developed in a way which allows for using them in different software configurations. It is possible to use them also outside of Protégé. The two factors that support this feature are: use of standard OWL API and component design with use of OSGi framework.

In fact different configurations, i.e. collaboration between standard (non context-aware) and SIM tools, were analyzed during very early stages of development. The comparison of various configurations is presented in Table 1 (see next page). In the *Ontology* column it is specified if user works with standard ontology or SIM knowledge base. In the *Editor* column it is specified if user works with Protégé without extensions (Standard) or works with Protégé with SIM plug-in (Context-aware). The next column *Reasoner* defines if user works with standard reasoner (Standard; e.g. Pellet [9]) or Conglo-S reasoner (Context-aware). The last column *Comment* contains remarks on a specific configuration.

The set of SIM tools works correctly in all the enlisted configurations. It was achieved by integration with OWL API, especially by the exploit of flow of events provided by *OWLOntologyManager*.

Table 1. Comparison of various configurations of collaboration between standard and context-aware tools

No.	*Ontology*	*Protégé*	*Reasoner*	*Comment*
1	Standard	Standard	Standard	-
2	Standard	Standard	Context-aware	Context-aware reasoner can cooperate with standard ontologies. It works analogically to the first configuration.
3	Standard	Context-aware	Standard	Standard ontology can be edited by context-aware plug-in, analogically as for non-extended Protégé.
4	Standard	Context-aware	Context-aware	Works analogically to the second and third configurations.
5	Context-aware	Standard	Standard	Context-aware ontology can be edited in Protégé only by expert users, familiar with ontological description of the metamodel (see Sec. 2.3 and 2.5). Reasoner provides sound (but not complete) conclusions.
6	Context-aware	Standard	Context-aware	Context-aware ontology can be edited in Protégé only by expert users familiar with ontological description of the metamodel (see Sec. 2.3 and 2.5). Reasoner provides sound and complete conclusions.
7	Context-aware	Context-aware	Standard	Context-aware ontology is edited with use of designated views. Reasoner provides sound conclusions but not complete ones.
8	Context-aware	Context-aware	Context-aware	Complete set of context-aware tools

OWLOntologyManager is a standard OWL API interface so, as it was previously mentioned before, we can also integrate SIM tools into systems that are devoid of Protégé components. An example of such a system is a Conglo-S console, which function as a demonstrator of tarset algebra for users, and allows for formulating commands in dedicated query/command language developed in ANTLR [13] (cf. Fig. 7).

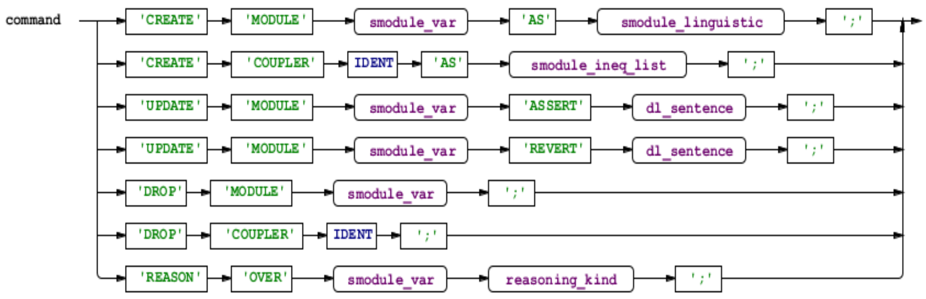


Fig. 7. Grammar for console command in Conglo-S (from ANTLR-works tool [14])

Further possibilities of interoperability stem from using OSGi component-based framework. This decision allows us to easily embed the tools into any system exploiting this standard (including Eclipse RCP) and is important for the further plans of deployment of the tools into SYNAT project YADDA platform [16].

4 Possible Applications for Cultural Heritage

This section contains a discussion about possible applications of the introduced ideas within the SyNaT system designed for facilitation of management and use of digital objects connected with cultural heritage.

4.1 Improving the Efficiency of Reasoning

Let us consider use of SIM model for improving the efficiency of inference from a knowledge base organized in accordance with CIDOC CRM. The motivation behind this work is strictly connected with work (see [15]) of PCSS (Poznan Supercomputing and Networking Centre; a partner in SyNaT). PCSS prepared a rich source of linked data on the basis of the contents of D-Libra system: a system designated for indexing library resources in digital form.

PCSS puts a great effort into adjusting the data in compliance with the requirements of CIDOC. The resulting base is very large and can be processed only by reasoners performing variants of *pD** inference, like OWLIM. Such inference is attractive because of computational properties, but it is not as exhaustive as standard DL reasoning.

Loading the base in its original form to a standard reasoner is an infeasible task. Tests performed by us indicate that for a terminology of such complexity an efficiency barrier occurs at about 3000 individuals. Because the base is several orders of magnitude larger, modularization seems to be a viable solution.

We propose to introduce two-dimensional contextual breakdown of the structure of the base. A "vertical" breakdown would consists in separating the terminology into conceptual units embracing: time-space information, events, agents, and objects. A "horizontal" breakdown would concern the description of the world (context instances) and would consist in separating information objects coming from different libraries.

We plan to perform experiments in order to answer the question whether such decomposition would be enough to perform full OWL reasoning (with Open World Assumption), and what would be the cost of combining knowledge from different contexts in order to answer more comprehensive queries.

4.2 Introduction of Contextual Reasoning to Library Catalogs

Another very interesting stream of our work on contextualization of the SyNaT knowledge base is connected with library catalogs. Books (and other digital objects) are currently organized in accordance with a special catalog of terms (hierarchical thesaurus) called KABA. Use of KABA gives great advantages in unifying library information records, but it also has some disadvantages.

KABA in the present form cannot be treated as a viable source of terminological information for inference engines. One of the reasons is the fact that it carries only loose relationships between terms. KABA distinguishes between broader and narrower words and expression, but this hierarchy should not be confused with concept taxonomy. For example, *a lighthouse* has a broader term *rescue missions*, while naturally there is no subsumption between the terms. However, to some extent, we may say that a book about lighthouses carries some knowledge potentially useful for carrying out rescue missions.

Our aim here is to introduce contextual method of knowledge base organization to distinguish different fragments of KABA hierarchy, and to determine which parts of it and under what assumptions may be treated as proper taxonomies. This is an ambitious task because most probably it would require us to exploit some kind of meta-description (in order to differentiate between various parts of KABA) and to deploy a method of converting parts of PCSS knowledge base ABox containing KABA terms to context types.

While the difficulty of the task seems to be vast, the potential advantages are very appealing. KABA constitutes a large portion of PCSS knowledge base (close to the number of 300,000 individuals), and using it for improving semantic search seems very natural. Meta-knowledge about how to relate different categories of queries to different portions of the hierarchy would greatly facilitate and improve the process of answering. Moreover, the specific fragments of KABA may be used as starting-point ontologies for describing various fields of scientific knowledge. Furthermore, out of the necessity, KABA has to be maintained on the current basis (as it is used for indexing new papers, journals, and books) and should keep pace with the development of science and technology, so that such ontologies may be automatically or semi-automatically kept up-to-date.

5 Related and Further Work

The approach presented in the paper is not the only one concerning knowledge base modularity. In fact there are several notable methods in the field, with probably most prominent $\mathcal{E}$-Connections [17] and Distributed Description Logics [18]. In the both approaches authors take as a starting point a pair of ontologies and relate their terms with use of special syntactic constructions: axiom with linking relations in the case of $\mathcal{E}$-Connections, and bridge rules in the case of DDL.

Another very significant stream of works connected with modularity is the research concerning conservative extensions ([5], [19]). While potential applications of the theory seem to be very broad, the most notable studies in the area concern ontology evolution (preparing new versions and assessing their influence on possible prior extensions), and improvement of reasoning performance (with use of automated division of an ontology into modules).

Also within the research area there were formulated proposals of algebras for ontologies. One of the examples is described in [20]; without going into details, the authors postulate treatment of ontologies as graphs and try to use an algebra of graphs including mainly set-theoretic operations (like intersection, union etc.).

Summarizing the above short analysis we should clearly state that probably every element of the proposed here framework can be related to ongoing research. While we are aware of importance of such comparison (and, in fact, performed it, e.g. in [21]), we stress the fact that we primarily perceive the strength of the framework in its wholeness and uniformity. We introduce relatively simple mathematical apparatus and consequently build over it additional elements, driven by the idea that modularity should be considered as one of the fundamental factors in ontology engineering. Through such course of work we managed to integrate several dissipated areas of interest (like ontology evolution, integration, querying, and design) into one consequential stream of works, which would hopefully lead to creation of a model and a suite of tools for knowledge bases comparable in maturity and flexibility to relational databases.

The holistic framework presented in this chapter, although mature from the theoretical point of view, is currently under development. Our work concentrates on preparing a basic set of tools, which would allow us go through a process of designing a tarset knowledge base, as a proof concept for our approach and as an example of practical application of presented theories.

To sum up: Our main goal is to create a set of conceptual and development tools that will make development of knowledge-based systems as natural for a software engineer as it is now in the case of data-based systems. This follows the idea that we call “New Knowledge Engineering Initiative” (NKEI) that we promote among researchers and practitioners. To this end, we, among others, plan to implement a subset of KQL [22] as a suitable and semantically adequate language for querying modularized knowledge bases. In parallel, further theoretical work will be conducted. Among others, we plan to investigate other types of dependencies between tarsets than those that support context-aware knowledge-bases. This would enable us to construct modularized knowledge bases with definable modalities, according to the needs of complicated reality that we live in and that knowledge engineering must face.

References

1. Goczyla, K., Waloszek, A., Waloszek, W.: S-modules - Approach to Capture Semantics of Modularized DL Knowledge Bases. In: Proc. of KEOD 2009, pp. 117–122 (2009)
2. Goczyla, K., Waloszek, A., Waloszek, W.: A Semantic Algebra for Modularized Description Logics Knowledge Bases. In: Proc. of DL 2009. CEUR-WS, vol. 477 (2009)
3. Goczyła, K., Waloszek, A., Waloszek, W.: Algebra of Ontology Modules for Semantic Agents. In: Nguyen, N.T., Kowalczyk, R., Chen, S.-M. (eds.) ICCCI 2009. LNCS(LNAI), vol. 5796, pp. 492–503. Springer, Heidelberg (2009)
4. Goczyła, K., Waloszek, A., Waloszek, W., Zawadzka, T.: Modularized Knowledge Bases Using Contexts, Conglomerates and a Query Language. In: Bembenik, R., Skonieczny, L., Rybiński, H., Niezgodka, M. (eds.) Intelligent Tools for Building a Scient. Info. Plat. SCI, vol. 390, pp. 179–201. Springer, Heidelberg (2012)
5. Lutz, C., Walther, D., Wolter, F.: Conservative extensions in expressive description logics. In: Proc. of IJCAI 2007, pp. 453–459 (2007)
6. Goczyla, K., Waloszek, A., Waloszek, W.: Contextualization of a DL knowledge base. In: Proc. of DL 2007, pp. 291–299 (2007)
7. Goczyla, K., Waloszek, A., Waloszek, W.: Hierarchical partitioning of ontology space into contexts (in Polish). In: Kozielski, S., Małysiak, B., Kasprowski, P., Mrozek, D. (eds.) Bazy Danych. Nowe Technologie - Architektura, Metody Formalne i Zaawansowana Analiza Danych, pp. 247–260. Wydawnictwa Komunikacji i Łączności, Warszawa (2007)
8. Waloszek, A.: Hierarchical contextualization of Knowledge Bases (in Polish). Doctoral dissertation, Gdańsk University of Technology (2010)
9. Pellet: OWL 2 Reasoner for Java, `http://clarkparsia.com/pellet`
10. Goczyla, K., Waloszek, A., Waloszek, W.: A Semantic Algebra for Modularized Description Logics Knowledge Bases. In: Proc. of DL 2009, pp. 1–12 (2009)
11. The OWL API, `http://owlapi.sourceforge.net`
12. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A Practical OWL-DL Reasoner. Journal of Web Semantics 5(2), 51–53 (2007)
13. Parr, T.: The Definitive ANTLR Reference: Building Domain-Specific Languages. Pragmatic Bookshelf (2007)
14. Bovet, J.: ANTLR Works: The ANTLR GUI Development Environment, `http://www.antrl.org/works` (accessed June 11, 2012)
15. Mazurek, C., Sielski, K., Stroiński, M., Walkowska, J., Werla, M., Węglarz, J.: Transforming a Flat Metadata Schema to a Semantic Web Ontology: The Polish Digital Libraries Federation and CIDOC CRM Case Study. In: Bembenik, R., Skonieczny, L., Rybiński, H., Niezgodka, M. (eds.) Intelligent Tools for Building a Scient. Info. Plat. SCI, vol. 390, pp. 153–177. Springer, Heidelberg (2012)
16. About YADDA, `http://yaddainfo.icm.edu.pl` (accessed June 11, 2012)

17. Kutz, O., Lutz, C., Wolter, F., Zakharyaschev, M.: E-connections of abstract description systems. Artificial Intelligence 156(1), 1–73 (2004)
18. Borgida, A., Serafini, L.: Distributed Description Logics: Assimilating Information from Peer Sources. In: Spaccapietra, S., March, S., Aberer, K. (eds.) Journal on Data Semantics I. LNCS, vol. 2800, pp. 153–184. Springer, Heidelberg (2003)
19. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Modular Reuse of Ontologies: Theory and Practice. J. of Artificial Intelligence Research (JAIR) 31, 273–318 (2008)
20. Mitra, P., Wiederhold, G.: An Ontology-Composition Algebra. In: Handbook on Ontologies, pp. 171–216. Springer (2004)
21. Goczyla, K., Waloszek, A., Waloszek, W., Zawadzka, T.: Analysis of Mapping within S-module Framework. In: International Conference on Knowledge Engineering and Ontology Development, KEOD 2011 (2011)
22. Goczyła, K., Piotrowski, P., Waloszek, A., Waloszek, W., Zawadzka, T.: Terminological and Assertional Queries in KQL Knowledge Access Language. In: Pan, J.-S., Chen, S.-M., Nguyen, N.T. (eds.) ICCCI 2010, Part III. LNCS(LNAI), vol. 6423, pp. 102–111. Springer, Heidelberg (2010)