Integrating SUMO and OMDoc

Guided Research: Final Report

Dimitar Misev
Supervisor: Prof. Dr. Michael Kohlhase

Jacobs University

May 2010

Abstract

Ontologies promote and facilitate interoperability among information systems, intelligent processing by agents and sharing and reuse of knowledge among systems. They have recently become popular with the advent of the Semantic Web, as an appropriate tool for developing a common reference terminology and language in specific application domains. The Suggested Upper Merged Ontology (SUMO) and its domain ontologies form one of the largest formal public ontologies in existence today. OMDoc is a markup format and data model for Open Mathematical Documents, that serves as a semantics-oriented representation format for mathematical knowledge. SUMO would benefit from making use of the modularity and documentation infrastructure that OMDoc offers, whereas OMDoc would benefit from having access to such a large formal ontology. In this project we have started an integration between the two projects, forming a solid foundation on which to base further collaboration.

1 Introduction

Knowledge representation is the study of how knowledge about the world can be represented and how to reason about that knowledge. Knowledge Engineering is "the task of extracting knowledge from a human knowledge-holder and incorporating that knowledge into computer programs" [16]. Knowledge engineers perform this task by building knowledge-based systems, using ontologies as the most general component to organize terms, concepts, and relationships that represent a field or domain.

Representing complex domains concentrating on general, abstract concepts is called ontological engineering. Ontology engineering is related to knowledge engineering, but operates on a grander scale [19]. An ontology consists of a set of concepts, axioms, and relationships that describe a domain of interest. An upper ontology is limited to concepts that are meta,

generic, abstract and philosophical, and therefore are general enough to address (at a high level) a broad range of domain areas. Ontologies have recently become popular with the advent of the Semantic Web, a semantic-aware version of the World Wide Web, as an appropriate tool for developing a common reference terminology and language in specific application domains.

Knowledge Management (KM) refers to a multi-disciplined approach to achieving organizational objectives by making the best use of knowledge. KM focuses on processes such as acquiring, creating and sharing knowledge and the cultural and technical foundations that support them.

This project aims at integrating two projects closely related with these concepts – the Suggested Upper Merged Ontology (SUMO), and OMDoc, a markup format and data model for Open Mathematical Documents. We look in more detail at these projects in the next two sections.

1.1 SUMO

The Suggested Upper Merged Ontology (SUMO) [18] and its domain ontologies form one of the largest formal public ontologies in existence today[1] (with 20,000 terms and 70,000 axioms). It is developed by the IEEE Standard Upper Ontology Working Group, with the goal to develop a standard ontology that will promote data interoperability, information search and retrieval, automated inferencing, and natural language processing.

The SUMO itself is an ontology that consists of several interdependent sections [22]: Structural Ontology, with definitions for relations that serve as the framework for defining the ontology; Base Ontology, containing very fundamental notions such as the distinction between objects and processes; Set/Class Theory Ontology; Numeric section with basic arithmetic functions; section on temporal relations; Mereotopology Ontology, containing part/whole relations; Graph Theory section; and finally a Unit of Measure section, which provides definitions of the SI and other unit systems. Many domain ontologies which specify the concepts and axiomatic content of a particular domain have been additionally developed, e.g., ontologies of geography, distributed computing, finance, transportation, etc. Finally, there is the MILO (Mid-Level Ontology), an ontology developed as a bridge between the abstract content of the SUMO and the rich detail of the domain ontologies.

SUMO is written in a variant of the Knowledge Interchange Format (KIF) language [7] called Standard Upper Ontology Knowledge Interchange Format (SUO-KIF) [20]. SUO-KIF is a language designed for use in authoring and interchange of knowledge. It is logically comprehensive, providing for the expression of arbitrary logical sentences. Technically SUO-KIF

[1] Another open source ontology of comparable size is OpenCyc.

is a higher-order language, as it allows variables in the predicate position, e.g. `(?REL ?A ?B)`. In a practical reasoning system however, `?REL` needs to only range over the set of relations already defined in the knowledge base, which is first-order. Most reasoning systems take the broader interpretation that `?REL` ranges over the universe of all possible relations and disallow variables in the predicate position. For this reason, in SigmaKEE all such statements are pre-processed by adding a "dummy" relation, such as `(holds ?REL ?A ?B)`, and then removing it in the proof output [21].

1.2 OMDoc

OMDoc [11] is a markup format and data model for Open Mathematical Documents. It serves as semantics-oriented representation format for mathematical knowledge. The OMDoc format aims to

1. be ontologically uncommitted so that it serves as an integration format for mathematical software systems;
2. provide a representation format for mathematical documents that combine formal and informal views of all the mathematical knowledge contained in them;
3. be based on structural/content markup to guarantee both 1. and 2;
4. support scalability and web-integration.

1.3 Statement and Motivation of Research

The goal of this project is to start an integration between SUMO and OMDoc, providing a solid foundation on which to base further collaboration. SUMO is a comprehensive and quite stable[2], richly axiomatized ontology, based on a simple and clean language. It does have a couple of problems however:

- SUMO does not have any modularity infrastructure. If we would want to use the `Transportation` domain ontology for example, we would have to work with all of SUMO, because it is not possible to determine which other ontologies exactly it is built on. Modularization is a very important issue, as it directly affects the scalability of an ontology.
- It lacks a good documentation infrastructure. Objects can be documented with the `documentation` relation, but what if we would want to document an axiom for example? There is not much choice beyond a comment in the code.
- It is not particularly compliant with the Semantic Web technologies[3].

[2] Its structure has not changed much in the last couple of years.
[3] But there is a SUO-KIF↔OWL translator that already addresses this.

We address these problems by translating SUMO to OMDoc. OMDoc has been proven suitable as a translation target to and from other languages; in fact, one of its original goals was to provide an exchange language for formalized mathematics. With its flexible mechanisms for importing theories it supports modularity and scalability. OMDoc is especially suitable for documenting ontologies as we can flexibly mix and interlink mathematical formulae with natural language, and comes with an elaborate, adaptive presentation framework for creating human-readable documents from semantic markup. We present a tool that supports this integration by automating the process of translating between SUO-KIF and OMDoc. Additionally, we show how a TNTBase repository with SUMO in OMDoc format can help ontology engineers to further refine and adapt the ontology engineering process to their needs. We do not see TNTBase as replacing but rather complementing existing ontology engineering tools, so we have integrated it into SigmaKEE, in order to make its usage transparent to the users.

1.4 Related Work

Most related to this project is the effort of integrating OMDoc and RDFS/OWL. The semantic web ontology languages RDFS [9] and OWL [8] are widely used but limited in both their expressivity and their support for modularity and integrated documentation. In [14] it was proposed embedding the language concepts into OMDoc to make use of its modularity and documentation infrastructure, and two tools have been developed to support this. The OMDoc→OWL translation, implemented as a module in the Krextor XML→RDF extraction framework [13], allows one to use semantic web reasoners on tractable/decidable fragments of ontologies in OMDoc. The `rdf2omdoc` tool [12] on the other hand implements the OWL→OMDoc translation, allowing to easily document existing web ontologies in RDFS or OWL using the OMDoc documentation infrastructure. Similarly, there is also a SUO-KIF↔OWL translation, which is implemented in SigmaKEE.

The translation of the Mizar Mathematical Library (MML) [26] to OMDoc [1, 6, 17] is also related in that the Mizar language, similarly to SUO-KIF is first-order in style, however it is not designed for ontologies and the formalized mathematical content of the MML is quite different from the ontological data in SUMO.

1.5 Outline

In this section we have given an introduction to the problem, stated the motivation behind this project and its goal, and presented work related to ours. Next, we provide an introduction to the systems used in the later sections, namely SigmaKEE, TNTBase and JOMDoc. Section 3 documents all of the work that has been completed in the course of this project. Then in

Section 4 we evaluate the results from our work, and finally in Section 5 we conclude and give an outlook on possible improvements and future work.

2 Technology Overview

Here we introduce the software systems inherent to this project, and technical terminology that is used later on in this paper.

2.1 SigmaKEE

The Sigma Knowledge Engineering Environment (SigmaKEE) [20] is a Knowledge Based Reasoning (KBR) environment for developing and using logical theories. It was created to support the development of the Suggested Upper Merged Ontology (SUMO). SigmaKEE runs as an Apache Tomcat service, providing a browser interface to the users. The main components are written in Java, and the user interface is generated by JSP (Java Server Pages). The SigmaKEE does not provide tools for editing ontologies; instead it serves as an addition to an ontology editor (any text editor), supporting the ontology engineering process with facilities for structured examination, project management and debugging.

The most important functionality that can be directly accessed from SigmaKEE's main page includes:

Manifest Ontologies are organized in knowledge bases, that consist of one or more constituent files. Constituent files are files uploaded by the user which contain SUO-KIF statements. An uploaded knowledge base (KB) is indexed for high performance browsing and searching [25]. The *Manifest* page shows the uploaded files contained in a knowledge base.

Browse The fundamental part of the interface is the *Browse* page, which given a term, displays the SUO-KIF statements in which it appears. The terms in the result statements are interlinked, and allow to further browse the statements in which they appear. For each statement, an English paraphrase of it is additionally shown. The language paraphrases are automatically generated from the `format` SUO-KIF statements in the KB, which serve as templates for paraphrasing SUO-KIF statements in a certain language.

Graph Given a relation, this page displays a graph of the terms connected by it.

Ask/Tell The Vampire theorem prover [23] is tightly integrated into SigmaKEE, adding support for logical inference over the current KB. This functionality is available in the *Ask/Tell* page, where a user can add

("Tell") new statements to the KB, or pose logical queries ("Ask") to the KB, in which case the inferred proof will be displayed as a result, with the proof steps numbered and justified. When the experimental Controlled English to Logic Translation (CELT) system is loaded, the user can also enter assertions and queries in English.

SystemOnTPTP This experimental functionality is accessible from the *Ask/Tell* page. It allows the user to pose queries to any of the several theorem provers accessible via a service at the University of Miami. When a query is submitted, both the query and the entire contents of the KB are translated to TPTP format and sent to the selected prover [25].

Since some parts of this project deal with directly modifying SigmaKEE's code, an analysis of its code including the most important classes is necessary in order to get better insight on how it works internally.

KBManager The main class for Sigma, manages all loaded in-memory knowledge bases (`KB` instances).

KB Each `KB` instance contains all the content of the constituent files of that particular KB, as well as instances of the Vampire and CELT processes.

KIF This class is used by `KB` for reading SUO-KIF statements from a given file into `Formula` objects, as well as writing them to a file.

Formula A `Formula` is a representation of a SUO-KIF statement, providing methods for operating on individual formulas, formatting for presentation, and pre-processing for use by the `Vampire` class.

Diagnostics Finds problems in a given KB.

LanguageFormatter Generate natural language from logic.

OWLTranslator SUO-KIF↔OWL translation.

WordNet Given an English term, this class returns the associated SUMO concepts.

2.2 TNTBase

TNTBase [27] is an open-source versioned XML database obtained by integrating the XML database Berkeley DB XML into the server-side part of the Subversion revision control system. The system is intended as a basis for collaborative editing and sharing XML-based documents. It integrates versioning and fragment access needed for fine-granular document content management. As described in [29], two different interfaces can be used to communicate with TNTBase:

1. An XML-enabled SVN interface (xSVN), which works in the same was as a normal SVN interface, except that commits containing ill-formed XML files are automatically aborted.

2. A RESTful interface providing functionality for fragment access of the versioned collection of documents, including querying (possibly previous revisions) with XQuery [4] and modifying XML content with XQuery Update [5], as well as Virtual Documents, which are the equivalent to views in relational databases.

2.3 JOMDoc

JOMDoc [10] is a Java API for OMDoc documents, which facilitates the parsing of OMDoc XML documents into a Java structure, allowing to manipulate them conveniently, and serialize the result back to XML. Additionally it provides several services on top of this functionality, including context-sensitive conversion of OMDoc documents containing Content MathML/OpenMath to XHTML + Presentation MathML, concretion and abstraction of documents, validation for constraints that can not be checked by validating against an XML schema, etc.

JOMDoc has been extensively used in the SUO-KIF↔OMDoc translation, more specifically in the implementation of the `sumo2omdoc` tool, and the resolving of circular imports.

3 Work Documentation

This section documents all the work that has been completed in the course of this project. We start with a theoretical overview of a mapping between the SUO-KIF and OMDoc constructs, and then the tool the implements this mapping and allows to automatically translate between the two formats is presented. Next, we talk about two major problems that we had to solve in order to have a complete and robust translation – the construction of imports relations and elimination of circular imports. Finally, we show how ontology engineers would benefit by using a TNTBase repository for storing SUMO ontologies in OMDoc format.

3.1 Translating SUMO to OMDoc

SUMO consists of various ontologies written in the SUO-KIF language. The first step towards the integration between SUMO and OMDoc is to provide a way to translate between SUO-KIF and OMDoc. Here I outline the mapping between the SUO-KIF and OMDoc concepts and how this translation is carried out in theory.

Semantically there are four types of constants in SUO-KIF[4]:

1. Object constants, used to denote individual objects.
2. Function constants, denote functions on those objects.
3. Relations are expressed by relation constants.
4. Logical constants which express conditions about the world, and are true or false.

SUO-KIF objects can be either classes or individuals. Relations and functions can be defined as instances of the class of all relations and the class of all functions. Terms can be combined into relational, logical and quantified sentences, in order to express facts about the world. Let us look at the following example[5] which defines the concept of `Byte`, in order to get a better understanding of SUO-KIF.

```
(instance Byte UnitOfInformation)
(documentation Byte EnglishLanguage "One &%Byte of information.
    A &%Byte is eight &%Bits.")
(=> (equal ?NUMBER (MultiplicationFn 1 ?NUMBER))
    (equal (MeasureFn ?NUMBER Byte)
           (MeasureFn (MultiplicationFn ?NUMBER 8) Bit)))
```

We distinguish the following constants in this example: `UnitOfInformation`, `Byte`, `Bit` and `EnglishLanguage` are object constants; `MultiplicationFn` is the multiplication function which returns the result of multiplying two numbers, and `MeasureFn` is a function that given a number n and a unit returns n units, so these are function constants; `instance` is a relation constant that denotes that an object belongs to a specific set or class, and `documentation` documents an object in a specific language; the logical constants here are `=>` for implication and `equal` for equality (other logical constants in SUO-KIF are `not`, `and`, `or` and `<=>`). Variables in SUO-KIF are words in which the first character is `?` or `@`[6], so `?NUMBER` is a variable. `Byte` is an individual and `UnitOfInformation` is a class, of which `Byte` is an instance. The first and the second statements are relational sentences, whereas the third is a logical sentence stating the rule that n `Byte`s equals to $8n$ `Bit`s.

OMDoc is a three-layered semantic markup language for mathematical knowledge, as described in [11]:

Objects comprise mathematical formulae, usually composed of symbols and represented in content markup, using OpenMath [2] or MathML [3].

Statements are assertions about objects, including definitions, theorems, proofs and examples.

[4]There is no distinction in syntax however.

[5]Taken from the SUMO ontology.

[6]Called "row" or "sequence" variables, as they can hold a variable number of arguments.

Theories are a collection of symbol declarations, that describe mathematical objects specific to the theory, and statements which state the laws about the properties of the theory. In other words, theories provide context for symbols and statements.

Similarly to the OMDoc↔RDFS/OWL mapping presented in [14], a one-to-one correspondence between the SUO-KIF constructs and the three levels of markup supported by OMDoc can be easily derived:

Ontologies A SUO-KIF ontology corresponds to an OMDoc theory.

Objects Classes and individuals (including relations and functions) are modelled as symbols. In case of an individual, a type element is added with the class of which an individual is an instance. When a `documentation` is specified for a concept, it is added to the metadata of the corresponding symbol as a `dc:description`. Object constants which occur in a `documentation`[7] are substituted with `term` elements.

Statements All other SUO-KIF statements are modelled as OMDoc axioms. In an axiom, a SUO-KIF rule is formally represented as an OpenMath expression.

Figure 1 illustrates these correspondences with a simple example. The generated `example` theory reuses concepts from three other theories: `suo-kif` defines the basic language constructs of SUO-KIF, like logical and comparison operators, quantifiers, and the root SUMO class, Entity; `Merge` is the SUMO itself as translated to OMDoc, and `Mid-level` is the MILO.

3.1.1 `sumo2omdoc`

According to the mapping introduced in the previous section, I have implemented the `sumo2omdoc` tool which automates the process of translating between the SUO-KIF and OMDoc formats. It is based on the JOMDoc and SigmaKEE Java libraries, so for the reason of easier integration it is also implemented in the Java programming language. This also means that the tool is cross-platform portable, and that it will work on every operating system for which Java Runtime Environment 1.5 is available. `sumo2omdoc` along with all of SUMO translated to OMDoc is publicly available at `http://alpha.tntbase.mathweb.org/repos/sumo/`. The `README` file contains a description of the directory structure and information on how to build the code.

`sumo2omdoc` is designed to be both developer-friendly, allowing for easy integration in other projects, and user-friendly, by providing a command-line user interface. The code is heavily documented with `Javadoc`, so I will

[7] Denoted by prefixing a word with `&%`, e.g., `&%Byte` and `&%Bits` in the previous example.

example.kif

```
(subclass Camera Device)

(documentation Camera EnglishLanguage
"A &%Device which is capable of &%Photographing.")

(=>
   (instance ?CAMERA Camera)
   (capability Photographing instrument ?CAMERA))

(instance SonyA200 Camera)
```

example.omdoc

```
<theory name="example">
 <imports from="Merge.omdoc#Merge"/>
 <imports from="suo-kif.omdoc#suo-kif"/>
 <imports from="Mid-level.omdoc#Mid-level"/>
 <symbol name="Camera">
  <metadata>
   <dc:description xml:lang="en">
    A <term cd="Merge" name="Device">Device
    </term>which is capable of <term cd="Mid-
    Level" name="Photographing">Photographing
    </term></dc:description>
  </metadata>
 </symbol>
 <axiom xml:id="Camera-subclass" for="Camera">
  <FMP>
   <om:OMOBJ>
    <om:OMA>
     <om:OMS name="subclass" cd="Merge"/>
     <om:OMS name="Camera" cd="example"/>
     <om:OMS name="Device" cd="Merge"/>
    </om:OMA>
   </om:OMOBJ>
  </FMP>
 </axiom>
 <axiom xml:id="Camera-rule" for="Camera">
  <FMP>
   <om:OMOBJ>
    <om:OMA>
     <om:OMS name="implies" cd="suo-kif"/>
     <om:OMA>
      <om:OMS name="instance" cd="Merge"/>
      <om:OMV name="?CAMERA"/>
      <om:OMS name="Camera" cd="example"/>
     </om:OMA>
     <om:OMA>
      <om:OMS name="capability" cd="Merge"/>
      <om:OMS name="Photographing"
              cd="Mid-level"/>
      <om:OMS name="instrument" cd="Merge"/>
      <om:OMV name="?CAMERA"/>
     </om:OMA>
    </om:OMA>
   </om:OMOBJ>
  </FMP>
 </axiom>
 <symbol name="SonyA200">
  <type><om:OMOBJ><OMS cd="example"
   name="Camera"/></om:OMOBJ></type>
 </symbol>
</theory>
```

Figure 1: An example of mapping between SUO-KIF and OMDoc

omit the implementation details here. The command-line interface exposes all functionality to the user, and can be accessed through the according Windows or Linux/Mac scripts. For example, an ontology `Example.kif` could be translated to an OMDoc document `Example.omdoc` by executing `./sumo2omdoc --sumo-to-omdoc Example.kif -o Example.omdoc`. A list of all available command-line options along with descriptions of them can be printed with the `--help` option.

3.1.2 Identifying Imports Relations

In OMDoc, if a theory reuses concepts defined in another theory, it must explicitly import the other theory. While implementing the SUO-KIF to OMDoc translation, constructing the imports was a major problem, as SUMO does not have any explicit notion of imports. Given only one SUMO ontology and an undefined object in it, it is impossible to determine in which ontology is this object defined. Some ontologies specify imports relations as comments, but this is not respected in all ontologies and the translation can not rely solely on it. `sumo2omdoc` tries to handle this problem by first caching the terms defined in all of the ontologies that comprise SUMO, and then reusing this knowledge to construct the right imports when translating from SUMO to OMDoc.

3.1.3 Resolving Circular Imports

Many SUMO ontologies depend on each other in a circular fashion, e.g., an ontology O_1 uses concepts defined in O_2, and vice versa, O_2 uses concepts from O_1. Circular dependencies are considered a software design anti-pattern, for several reasons:

- There is a tight coupling of the involved modules, which reduces or makes impossible the separate re-use and testing of a single module.
- They can cause a domino effect, when a small local change in one module spreads over other modules causing unwanted global effects, as well as infinite recursions and other unexpected failures.
- Higher system complexity with respect to connections of knowledge – in a hierarchical system of N modules the complexity is bounded by $O((N(N-1))/2)$, whereas in a system containing circular dependencies this complexity is $O(N^2)$.

This is no exception in OMDoc, so we would want to avoid the circular imports that come up in the translation. One way to handle circular imports, which I initially implemented in `sumo2omdoc`, is to use local imports. Local imports only import the target theory, without further importing the theories imported by the target theory. `sumo2omdoc` recursively computes all the required imports and adds them to the result as local imports. This is not a particularly good solution because it introduces harder maintainability, e.g., suppose the imports in a theory τ have changed, then these changes must be propagated to all theories that import τ in this way. Moreover, local imports will become obsolete in later versions of OMDoc 1.3.

Circular imports can not be resolved by only reordering the direct imports between theories (a trivial example would be two theories importing on each other), and require moving concepts from one theory to another.

The method we describe here is based on refactoring common concepts to a new "utility" theory. Given a set of N OMDoc documents, we could represent them with two related graphs:

- $G_I(V_I, E_I)$ in which for each theory τ_v in the given documents, there is a vertex $v \in V_I$, and each edge $(u, v) \in E_I$ means that theory τ_u imports theory τ_v. Thus, G_I is a graph of the imports relations between the given OMDoc theories.

- $G_D(V_D, E_D)$ in which the vertices are the theory constitutive elements in the given theories (symbols, axioms and definitions), and an edge $(u_i, v_j) \in E_D$ means that u from theory τ_i depends on v from τ_j. G_D therefore represents the dependencies of individual theory parts. We additionally define $SG_i(V_{Di}, E_{Di})$ as a subgraph of G_D, which includes all vertices in V_D that belong to theory τ_i, and by the definition of a subgraph all edges from E_D that connect the vertices in V_{Di}.

Eliminating the circular imports in a set of OMDoc documents is equivalent to breaking the cycles in G_I, i.e. transforming it to a Directed Acyclic Graph (DAG). This could be done by running Algorithm 1 on G_I. Identifying the

Algorithm 1 break_cycles(G_I, G_D)

```
while true do
  c := identify_cycle(G_I)
  if c = ∅ then
    return
  end if
  break_cycle(c, G_I, G_D)
end while
```

circular imports is relatively easy, and can be done by running a Depth First Search (DFS) algorithm modified to detect cycles on G_I. The most important part of the algorithm is `break_cycle`(c, G_I, G_D) (outlined in Algorithm 2). Given a cycle $c = v \rightsquigarrow \ldots \rightsquigarrow u \rightsquigarrow v$, it rearranges the vertices G_I that are part of c, so that they are no longer circularly dependent on each other, while preserving the original dependency relations between them in G_D. The algorithm starts by choosing one edge (u, v) from c where the cycle will be broken, removes it and adds a new vertex v_{uv} to V_I along with a new edge (u, v_{uv}) to E_I. Now c is not a cycle any more, as it has been transformed from $v \rightsquigarrow \ldots \rightsquigarrow u \rightsquigarrow v$ to $v \rightsquigarrow \ldots \rightsquigarrow u \rightsquigarrow v_{uv}$. Then it moves to τ_{uv} all parts in theory τ_v on which parts in τ_u depend (set V_v), as well as any vertices that are part of the cycle and on which each vertex in V_v depends directly or indirectly (set V_w), by renaming the corresponding vertices in G_D and updating the imports in G_I.

Figure 2 shows an example of breaking one edge. There are three documents – A, B and C, in which $a, b, \ldots, h, i$ are concepts, and x, y and z

Algorithm 2 break_cycle(c, G_I, G_D)

$(u, v) := choose\_edge(c)$
$V_v := \{v | u \in V_{Du} \wedge v \in V_{Dv} \wedge (u, v) \in E_D\}$ // concepts in v used by u
if $V_v = \emptyset$ **then**
  return
end if
$V_I := V_I \cup \{v_{uv}\}$ // create a new theory
// u imports the new theory now, instead of v
$E_I := (E_I \setminus \{(u, v)\}) \cup \{(u, v_{uv})\}$
// get all concepts on the cycle
$V_C := \{y | x \in V_I \wedge contains(c, x) \wedge y \in V_{Dx}\}$
for $w_v \in V_v$ **do**
  // V_w denotes all concepts that will be moved to v_{uv} – w_v itself, and
  // all concepts on the cycle on which it depends directly or indirectly
  $V_w := \{w_v\} \cup \{x | w_v \rightsquigarrow \ldots \rightsquigarrow x \wedge x \in V_C\}$
  for $w_i \in V_w$ **do**
    $V_d := \{x | x \in G_D \wedge (x, w_i) \in E_D\}$ // all parts that depend on w_i
    for $w_d \in V_d$ **do**
      // w_d depends on w_i, but w_i will be moved to v_{uv}, so an import
      // from d to v_{uv} is added ($d \in V_I$ corresponds to $V_{D w_d}$)
      $E_I := E_I \cup \{d, v_{uv}\}$
    end for
    $w_i := w_{uv}$ // move w_i to theory τ_{uv}
  end for
end for

are axioms. In the context of the two graphs introduced earlier, A, B and C make up V_I and the thick arrows between them are the edges in E_I (an arrow from A to B means that A imports B). The concepts and axioms, along with the thin arrows between them are part of G_D. In the example we assume the following scenario: a cycle $A \rightsquigarrow B \rightsquigarrow C \rightsquigarrow A$ has been identified, and using the strategy of choosing the first edge, the cycle is broken at $A \rightsquigarrow B$. After this the algorithm terminates, as there are no more cycles left in the graph.

It is easy to see that the algorithm works correctly. When breaking a cycle c at an edge (u, v), the edge is removed from c and a new vertex v_{uv} is introduced which does not depend on any vertex on the cycle, and on which at least u depends. So we can conclude that `break_cycle` breaks a given cycle, but does not introduce any new cycles in the graph, therefore `break_cycles` removes all cycles from the graph in finite time. More specifically, its time complexity is roughly[8] $O(KC(|V_I| + |E_I|))$, where C is the

[8]It is hard to estimate an exact bound, since its behaviour largely depends on the strategy used to choose an edge for breaking from the cycle.

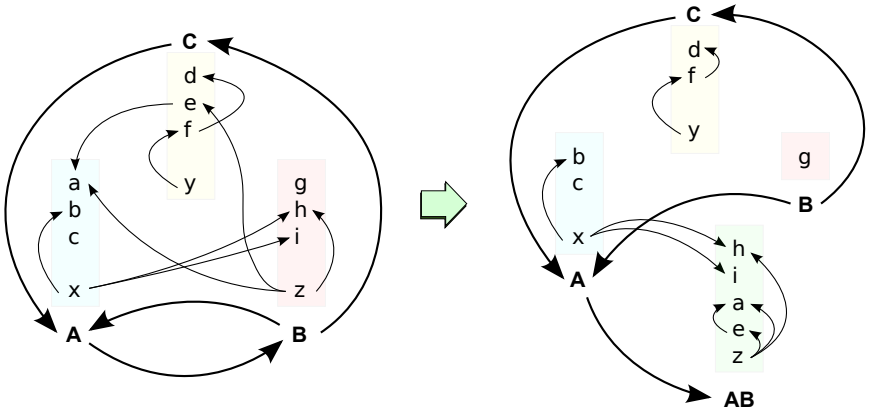

Figure 2: An example of resolving circular imports

number of identified cycles and K is some polynomial bound of `break_cycle`.

The algorithm is not flawless however, and problems could arise in certain cases. Since it involves moving of concepts between theories and renaming them in cases of name clashes, all theories that depend on these concepts need to be accordingly updated. This is not a problem in our case because all of SUMO is available when running the algorithm, but often it will not be the case.

An implementation of the algorithm is available in the JOMDoc library (in the `org.omdoc.jomdoc.transform.CircularImportsResolver` class). This functionality can be used by specifying the `--resolve-circular` option in `sumo2omdoc`, which takes a directory with OMDoc documents as an argument, and optionally the `--copy` option can be used to specify that the results should be copied in a different directory instead of overwriting the original documents.

3.2 TNTBase and SUMO

Ontology engineering is a discipline that is closely related to software engineering. Usually engineers work together on one project, thus support for collaborative development, integrated testing and evaluation is essential. Normally, a Subversion server or a similar revision control system would suffice for this purpose. In our project however, we have decided to use TNTBase, which is a versioned XML database. Since we have already translated the SUMO ontologies to OMDoc XML format, we can make use of the XML-specific functionality that TNTBase offers. Here we concentrate on Virtual Documents [29, 28], and how they could help ontology engineers to further refine and customize the ontology engineering process to their needs, beyond what tools with static functionality like SigmaKEE

already offer. VDs allow to integrate XQuery-based computational facilities into XML documents. They are "XML database views" analogous to views in relational databases, only that in TNTBase VDs are the result of XQueries computed over the XML files in the TNTBase file-system, instead of SQL queries computed over the database tables. Internally VDs are quite different from usual documents in the repository, but otherwise they appear as normal files in the TNTBase file-system, that the user can browse, validate, query and transparently modify.

In [15] it was presented how Virtual Documents can be utilized in the refactoring of ontologies, with examples for renaming entities, splitting or merging modules, rewriting axioms, lowering expressivity and stripping axiom annotations. Even though this paper is with focus on ontologies written in the OWL 2 Web Ontology Language, the presented use cases are still relevant in our case and could be easily adapted to SUMO ontologies in OMDoc format. In this paper we illustrate how VDs could be used to help the user focus on a particular subsection of a knowledge base. For example, if we would be working on something related to `Europe`, then there is no need to load the whole `CountriesAndRegions` ontology, but only the parts relevant to `Europe`. The VD in Appendix A allows us to do exactly this – given a document in the TNTBase repository and a particular class on which we want to concentrate, it computes a view of all its subclasses and instances, and related axioms.

In order to make TNTBase easily accessible to SUMO users we have added minimal but very essential support for it in SigmaKEE – loading knowledge bases directly from a remote TNTBase repository. The user can set a remote repository URL on the `Preferences` page in SigmaKEE. Then, on the `Manifest` page, the user can also load a selected constituent from the previously specified repository besides loading SUO-KIF files from the file-system, (Figure 3). Virtual Documents in the repository can be also selected, which results in loading the "view" they compute on a certain ontology. Since the content in TNTBase is in OMDoc format, in order to be understood by SigmaKEE it is automatically translated to SUO-KIF format by `sumo2omdoc` after it has been retrieved from the repository. With this we have achieved seamless integration, as loading files in OMDoc format from a TNTBase repository is just as easy as loading constituents from the file-system.

4 Evaluation

I have evaluated the translation between SUO-KIF and OMDoc on all SUMO ontologies by quickly examining the results. For thoroughly testing the translation, a simple control ontology that contained all SUO-KIF constructs was used. The `sumo2omdoc` tool is non-destructive, i.e., given an on-

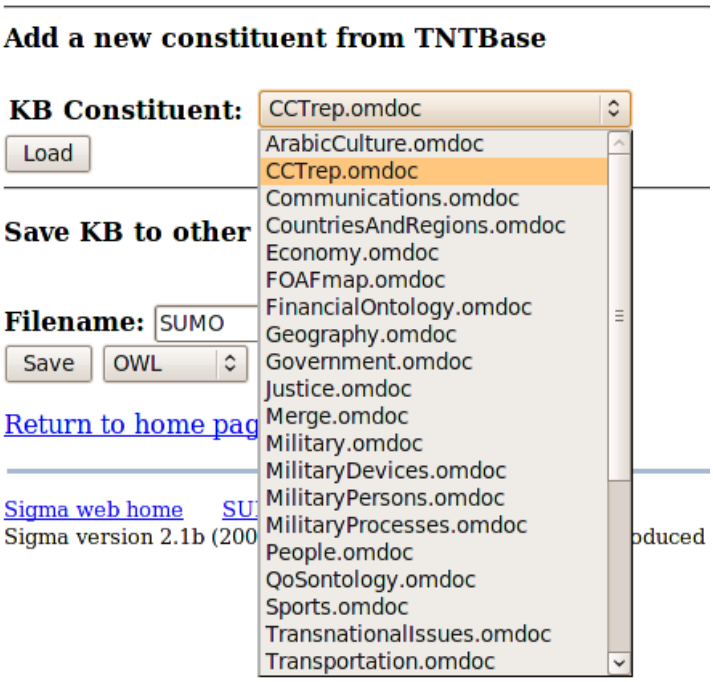

Figure 3: Loading a KB from TNTBase

tology O, it holds that $O = \text{OMDoc}{\rightarrow}\text{SUO-KIF}(\text{SUO-KIF}{\rightarrow}\text{OMDoc}(O))$. This has been verified with the `diff` Unix file comparison utility – for every SUMO ontology O, O' was computed by translating O to OMDoc format and translating this intermediate file back to SUO-KIF format, and then running `diff` O O' has only revealed differences in indentation and new lines. This is an important property of `sumo2omdoc` as it allows us to safely translate back and forth between SUMO and OMDoc, which is essential in scenarios like the previously discussed TNTBase integration, and it shows that the OMDoc to SUO-KIF translation is correct. However, this does not necessarily mean that the SUO-KIF to OMDoc translation is correct; the correctness of this direction was checked by successfully validating the generated OMDoc document against the OMDoc 1.3 RelaxNG schema. Additionally this is supported by the fact that the JOMDoc library, which is compatible to OMDoc 1.3, was used in the translation.

The circular imports resolving algorithm (cf. section 3.1.3) has been evaluated on a set of manually created documents that cover many specific cases. The extreme test for checking the performance and robustness of the algorithm consisted of running it over all SUMO ontologies[9], using a simple strategy of always choosing to break the first edge of a cycle. Starting with 36 documents, the algorithm eliminated 21 cycles, which involved moving

[9]The SUMO, the MILO, and the domain ontologies.

26086[10] symbols and axioms to newly created theories, resulting in 54[11] documents at the end. Changing to a more intelligent strategy of choosing an edge (u, v) such that v defines the least number of concepts with respect to all theories on the cycle, lead to a much faster completion of the algorithm as it involved moving only 4929 parts, but breaking more cycles (34), ending with 58 documents. The result from the algorithm has been verified with the imports validation functionality available in JOMDoc, which checks for symbol visibility, circular and redundant imports. JOMDoc has only reported redundant imports, which are not very critical. In order to solve this problem we will add an option to automatically fix inconsistencies such as redundant imports to JOMDoc, as currently it can only detect them.

5 Conclusion and Future Work

In this project a solid foundation for integration between SUMO, the Suggested Upper Merged Ontology, and OMDoc, a markup format and data model for Open Mathematical Documents, has been laid out. The SUMO ontologies are written in the SUO-KIF language, which is different from the OMDoc format, so we have first constructed a semantic-preserving mapping between the two formats. This enables SUMO to make use of OMDoc's modularity and documentation infrastructure. OMDoc users on the other hand will benefit from being able to use a large formal ontology. Based on the discussed mapping a `sumo2omdoc` tool has been implemented, which allows to automate the process of translating between SUMO and OMDoc. After translating SUMO to OMDoc, the explicit modularization of the ontologies has revealed that many of the ontologies are circularly dependent on each other. In order to handle this, an algorithm which automatically breaks circular imports in a set of OMDoc documents has been implemented in the JOMDoc Java library. With this the SUMO↔OMDoc translation is complete and provides a robust foundation on which to base further work. Finally, we have given some practical examples on how ontology engineers working on SUMO would benefit from the possibility to translate to OMDoc. More specifically, we have focused on TNTBase, showing how it can help ontology engineers to further refine and customize the ontology engineering process to their needs, beyond what tools with static functionality like SigmaKEE already offer. To make the usage of TNTBase as easy as possible, basic support for directly loading ontologies and "views" generated from Virtual Documents in OMDoc format has been added to SigmaKEE.

[10] Note that this does necessarily mean that 26086 *distinct* parts were moved, but that parts were moved so many times.

[11] There are less than $36 + 21 = 57$ theories, because it may happen that after breaking an edge a document could end up with no content at all, in cases when two documents are very tightly coupled.

It was beyond the scope of this project to develop something more than a low-level basis for collaboration between SUMO and OMDoc, so there is a lot of space for improvements and further work. One aspect that could be further improved is modularization of SUMO, in terms of partitioning of the big monolithic SUMO ontologies into smaller ones. Modularization techniques are considered to be a key capability in the current efforts towards scalable solutions that will enable ontologies to grow to the huge size that we can foresee in real world future applications [24]. Given the modular infrastructure that OMDoc provides, it would be relatively easy to implement an automated partitioning of big theories in JOMDoc based on some of the methods described in [24], e.g., partitioning based on the structural properties of an ontology.

In section 3.1.3 we presented an algorithm for eliminating circular imports. Even though this algorithm is with focus on OMDoc documents, it can be generalized to the dependencies between software modules/components instead of imports relations between theories, and applied in any other cases in software engineering where automated instead of manual elimination of circular dependencies would be preferred. A generic framework that only expects specifications on what is a module, and when one module depends on another, could be easily implemented and would allow to quickly adapt it to any case. The algorithm itself could be further optimized, by investigating what would be the optimal strategy for choosing which edge on a cycle to break. The performance of the algorithm is largely dependent on this strategy, as breaking a certain edge could lead to breaking many cycles at once, whereas another could involve moving thousands of concepts.

Further effort could be also put in the integration with TNTBase in order to make it even more transparent to the users. Ontology engineers would most likely edit the ontologies in SUO-KIF format instead of OMDoc, as it is more human-readable. Thus in their working copy of the remote TNTBase repository, the ontologies would be in SUO-KIF format, and the translation to OMDoc and back would be automatically done by `sumo2omdoc` on the server when committing or updating.

6 Acknowledgements

I would like to thank Michael Kohlhase, Christoph Lange, Florian Rabe and Vyacheslav Zholudev for the great guidance and direct support throughout this guided research. It would have been impossible without their help.

References

[1] Grzegorz Bancerek and Michael Kohlhase. Towards a Mizar Mathematical Library in OMDoc Format. In Roman Matuszewski and Anna Zalewska, editors, *From Insight to Proof – Festschrift in Honour of Andrzej Trybulec*, volume 10(23) of *Studies in Logic, Grammar, and Rhetoric*. The University of Bialystok, Polen, 2007.

[2] Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaetano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The Open Math Society, 2004.

[3] David Carlisle, Robert Miner, and Patrick Ion. Mathematical markup language (MathML) version 3.0. Candidate recommendation, W3C, December 2009. http://www.w3.org/TR/2009/CR-MathML3-20091215/.

[4] Don Chamberlin, Jonathan Robie, Daniela Florescu, Scott Boag, Jérôme Siméon, and Mary F. Fernández. XQuery 1.0: An XML query language. W3C recommendation, W3C, January 2007. http://www.w3.org/TR/2007/REC-xquery-20070123/.

[5] Don Chamberlin, Jonathan Robie, Daniela Florescu, Jim Melton, Jérôme Siméon, and Michael Dyck. XQuery update facility 1.0. Candidate recommendation, W3C, June 2009. http://www.w3.org/TR/2009/CR-xquery-update-10-20090609/.

[6] Elena Digor. An OMDoc representation of Mizar library. Bachelor's thesis, Computer Science, Jacobs University, Bremen, 2008.

[7] M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Stanford University, Stanford, CA, USA, 1992.

[8] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview. W3C recommendation, W3C, October 2009. http://www.w3.org/TR/2009/REC-owl2-overview-20091027/.

[9] Ramanathan V. Guha and Dan Brickley. RDF Vocabulary Description Language 1.0: RDF Schema. W3C recommendation, W3C, February 2004. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[10] JOMDoc project — Java library for OMDoc documents. http://jomdoc.omdoc.org, 2010. seen May.

[11] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, August 2006.

[12] Siarhei Kuryla. OMDoc as an ontology language: OWL→OMDoc translation implementation, 2009.

[13] Christoph Lange. Krextor – an extensible XML→RDF extraction framework. In Chris Bizer, Sören Auer, and Gunnar Aastrand Grimnes, editors, *Scripting and Development for the Semantic Web (SFSW2009)*, May 2009.

[14] Christoph Lange and Michael Kohlhase. A Mathematical Approach to Ontology Authoring and Documentation. In Jacques Carette, Lucas Dixon, Claudio Sacerdoti Coen, and Stephen M. Watt, editors, *MKM/-Calculemus 2009 Proceedings*, number 5625 in LNAI, pages 389–404. Springer Verlag, July 2009.

[15] Christoph Lange and Vyacheslav Zholudev. Previewing OWL Changes and Refactorings Using a Flexible XML Database. In Mathieu d'Aquin, Alexander García Castro, Christoph Lange, and Kim Viljanen, editors, *1st Workshop on Ontology Repositories and Editors*, CEUR Workshop Proceedings, Hersonissos, Greece, May 2010.

[16] A. Lopez and J. Donlon. Knowledge Engineering and Education. *Educational Technology*, 41(2):45–50, July 2001.

[17] Ankur Modi. Translating Mizar Mathematical Library to OMDoc. Bachelor's thesis, Computer Science, Jacobs University, Bremen, 2009.

[18] Ian Niles and Adam Pease. Towards a Standard Upper Ontology. In Chris Welty and Barry Smith, editors, *2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, pages 2–9, Ogunquit, Maine, USA, October 2001. ACM Press.

[19] Peter Norvig and Stuart Russel. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.

[20] Adam Pease. The Sigma Ontology Development Environment. In *Working Notes of the IJCAI-2003 Workshop on Ontology and Distributed Systems*, volume 71 of *CEUR Workshop Proceeding series*, Acapulco, Mexico, August 2003.

[21] Adam Pease. Standard Upper Ontology Knowledge Interchange Format. Language manual, available at http://sigmakee.cvs.sourceforge.net/*checkout*/sigmakee/sigma/suo-kif.pdf, 2009.

[22] Adam Pease, Ian Niles, and John Li. The Suggested Upper Merged Ontology: A Large Ontology for the Semantic Web and its Applications. In *In Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, page 2002, 2002.

[23] Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2,3):91–110, 2002.

[24] Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra, editors. *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*. Springer-Verlag, Berlin, Heidelberg, 2009.

[25] Steven Trac, Geoff Sutcliffe, and Adam Pease. Integration of the TPTPWorld into SigmaKEE. 2008.

[26] Andrzej Trybulec and Howard Blair. Computer assisted reasoning with MIZAR. In *IJCAI'85: Proceedings of the 9th international joint conference on Artificial intelligence*, pages 26–28, San Francisco, CA, USA, 1985. Morgan Kaufmann Publishers Inc.

[27] Vyacheslav Zholudev and Michael Kohlhase. TNTBase: a versioned storage for XML. In *Proceedings of Balisage: The Markup Conference 2009*, Balisage Series on Markup Technologies. Mulberry Technologies, Inc., 2009. available at http://kwarc.info/vzholudev/pubs/balisage.pdf.

[28] Vyacheslav Zholudev and Michael Kohlhase. Scripting Documents with XQuery: Virtual Documents in TNTBase. available at http://kwarc.info/kohlhase/papers/balisage10.pdf, 2010.

[29] Vyacheslav Zholudev, Michael Kohlhase, and Florian Rabe. A [insert XML Format] Database for [insert cool application]. In *Proceedings of XML Prague 2010*, 2010.

A Computing KB Subsections

Given a class name (`$name` in the code), and a specific document in the repository (`$file`), the XQuery code in Listing 1 computes a "view" that contains the parts of the document related to the class. Note that in order to keep the example simple and concise, we do not consider the imports relations.

Listing 1: Compute a subsection of a KB

```
declare default element namespace "http://omdoc.org/ns";
declare namespace om = "http://www.openmath.org/OpenMath";

(: return all subclasses of a class :)
declare function local:subclasses($d as element(omdoc),
  $name as xs:string) as element(symbol)* {
  let $names := $d//axiom/FMP/om:OMOBJ/om:OMA[
    child::om:OMS[1][@name='subclass'] and
    child::om:OMS[3][@name=$name]]/om:OMS[2]/@name/string()
  return
    for $n in $names return $d//symbol[@name=$n]
};
(: return all instances of a class :)
declare function local:instances($d as element(omdoc),
  $name as xs:string) as element(symbol)* {
  $d//symbol[child::type/om:OMOBJ/om:OMS[@name=$name]]
};
(: return all axioms in which a class/instance is used :)
declare function local:axioms($d as element(omdoc),
  $name as xs:string) as element(axiom)* {
  $d//axiom[descendant::om:OMS[@name=$name]]
};
(: compute a subsection of a KB $d :)
declare function local:subkb($d as element(omdoc),
  $s as element(symbol)) as element()* {
  let $name := $s/@name/string()
  let $subclasses := local:subclasses($d, $name)
  let $instances := local:instances($d, $name)
  let $axioms := local:axioms($d, $name)
  return $s | $axioms | (
    for $object in ($subclasses | $instances)
    return local:subkb($d, $object))
};

<theory name="{tnt:doc($file)/omdoc/theory/@name/string()}">{
  local:subkb(tnt:doc($file)/omdoc,
              tnt:doc($file)//symbol[@name = $class])
}</theory>
</omdoc>
```

In Listing 2 the corresponding VD Specification is shown. A VD Spec is a document template with XQuery inclusions, that TNTBase uses in order to figure out how to execute a particular XQuery and populate the query

results. In the VD Spec we set the parameters that the XQuery expects, namely `Europe` as a class for which we want to compute a subsection of the `CountriesAndRegions.omdoc` ontology.

Listing 2: VD Specification

```
<tnt:virtualdocument xmlns:tnt="http://tntbase.mathweb.org/ns">
  <tnt:skeleton xml:id="subkb">
    <omdoc xmlns="http://omdoc.org/ns"
      xmlns:om="http://www.openmath.org/OpenMath"
      xmlns:dc="http://purl.org/dc/elements/1.1/">
      <tnt:xqinclude query="tntbase:/docs/example/subkb.xq">
        <tnt:return><tnt:result/></tnt:return>
      </tnt:xqinclude>
    </omdoc>
  </tnt:skeleton>
  <tnt:params>
    <tnt:param name="class">
      <tnt:value>Camera</tnt:value>
    </tnt:param>
    <tnt:param name="file">
      <tnt:value>/docs/example/All.omdoc</tnt:value>
    </tnt:param>
  </tnt:params>
</tnt:virtualdocument>
```