e-Informatica Software Engineering Journal, Volume 10, Issue 1, 2016, pages: 51–67, DOI 10.5277/e-Inf160103

Automatic SUMO to UML translation

Bogumiła Hnatkowska[a]
[a]*Faculty of Computer Science and Management, Wrocław University of Science and Technology*
bogumila.hnatkowska@pwr.edu.pl

Abstract
Existing ontologies are a valuable source of domain knowledge. This knowledge could be extracted and reused to create domain models. The extraction process can be aided by tools that enable browsing ontology, marking interesting notions and automatic conversion of selected elements to other notations. The paper presents a tool that can be used for SUMO to UML translation. Such a transformation is feasible and results in a high-quality domain model which is consistent, correct, and complete, providing that input ontology has the same features.

Keywords: SUMO ontology, information retrieving, domain model, UML, class diagram

1. Introduction

A domain model is a key development artifact. It captures the most important types of objects in the context of the domain, i.e. entities that exist or events that transpire in the environment in which the system works [1,2]. The domain model, besides business object models and glossary [1], is used to document the domain to which the system relates. The domain model could be represented with the use of different notations, among which the most popular are Entity Relationships Diagrams and UML class diagrams.

Domain models should be of high quality to reduce the number of changes when the development proceeds. Among quality factors the most important are [3]: consistency, completeness, and correctness (3C).

Consistency and completeness could be perceived from 2 perspectives: external and internal, from which the external one is more difficult to achieve. External completeness means that we have identified all important entities and relationships in the domain, while external consistency means that we have documented the identified elements in a way that preserves their semantics [4]. On the other hand, the domain model is internally consistent when it contains no contradictions and it is internally complete when it does not include any undefined object, no information is left unstated or is to be determined [2].

The definition of model correctness is much vaguer. Some authors define it as a mixture of consistency and completeness [3], others [4] refer it to syntactic correctness (this meaning of correctness is used further in the paper).

A business analyst typically elaborates a domain model during a business modelling or requirement specification phase [2].

Different elicitation techniques serve to discover entities in the domain, e.g. interviews. However, the obtained results strongly depend on the complexity of the domain, business analyst experience and the quality of information sources. The more difficult domain, the less experienced an analyst or poor quality sources, the more likely worse quality of the resultant domain model.

On the other hand, domain knowledge is often included in existing ontologies and could be extracted from them. The extraction process could be (partially) automated, which would result in a high- quality domain model. Consistency and correctness of such a model could be guaranteed by construction, assuming that the source (ontology) itself is correct and consistent with the domain. The model completeness, at least the internal one, could also be checked.

Many papers prove that the domain knowledge represented by ontology can be widely used in the design process of information systems. For example, in [5] author analyses the role of ontologies in software engineering process. The author claims that ontology is a significant source of knowledge in the conceptualization phase and proposes the ontology life cycle as the background for a software development. A similar view is presented in [6] where the authors state that the integration with ontology can improve software modelling. An application of domain ontologies to conceptual model development is also in presented in [7].

There are many high-level ontologies currently developed, e.g. BFO, Cyc, GFO, SUMO. The last one, SUMO, seems to be very promising because it became the basis for the development of many specific domain ontologies. A particularly useful feature is that the notions of SUMO have formal definitions (expressed in SUO-KIF language) and at the same time they are mapped to the WordNet lexicon [8]. SUO-KIF is a variant of the KIF (Knowledge Interchange Format) language [9]. Knowledge is described declaratively as objects, functions, relations, and rules. SUMO and related ontologies form the largest formal public ontology in existence today [8, 9]. What is more, the ontologies that extend SUMO, are available under GNU General Public License.

The paper presents a tool for automatic SUMO to UML translation. It is thought as a support for a business analyst collaborating with business experts. The main functionalities include: browsing ontology content, selection of interesting elements, and translation of selected elements to a UML class diagram. The solution presentation covers the meta-model of SUMO notions (the main input to the transformation process), tool architecture and an example of the domain model which results from tool application. The genesis of the tool (related works) is also shortly described as well as the problems met during implementation, and the elements that will be included in the next release.

The only tool available on the Internet that supports SUMO is SUMO browser, called Sigma [10]. Tools that allow to create a UML class diagram from the existing ontology exist for other formalisms, e.g. OWL [11], but not for SUO-KIF. However, SUO-KIF could be translated to other formalisms, e.g. DLP [12].

SUMO was selected from existing ontologies for the following reasons:

– It constitutes the biggest set of ontologies which is freely available; SUMO contains definitions of more than 21 thousands of terms, and more than 70 thousands of axioms; moreover, the mapping of SUMO notion to WordNet is also available [9];
– SUO-KIF language is very flexible; it allows to handle relations among three or more things directly (e.g. OWL does not); it supports statements and rules written not only in First-Order Logic, but also (at least partially) in the Higher-Order Logic (e.g. "(believes John (likes Bob Sue))", when the second argument of "believes" is a proposition) [9];
– Existing translation of SUMO to OWL is a provisional and necessarily lossy [9], which calls into question its usefulness; on the other hand, it is possible to perform the reverse translation from OWL to SUMO, which seems more promising, because the result could be extended with the usage of SUO-KIF features;
– The flexibility of SUO-KIF is very similar to the SBVR standard [13], promoted by OMG, defining the meta-model for representation of business vocabulary, and business rules; SBVR statements could be directly translated either to SUO-KIF or UML.

UML was selected as the target language for translation because it is a general purpose modeling and specification language commonly used not only by programmers but also by business analysts. Besides, the Entity Relationship Diagram is a frequently selected notation to describe domain models. Together with OCL it forms a very useful tandem to define constraints on the domain behavior in a formal way. The UML class diagram could be easily translated to other representations, either more business oriented like SBVR (e.g. [13]) or more program oriented like Java, C++, SQL (e.g. [14]).

A tool for automatic SUMO to UML translation can be useful for anyone (especially a business analyst) who would like to familiarize with some specific domain. Theoretically, he or she can read the ontology definition for that purpose. Unfortunately, even if the SUMO browser is in use, knowledge extraction from SUMO is a challenge. SUMO is expressed in the textual SUO-KIF language which is not not commonly known. After a while, a reader is overloaded with textual definitions. The aim of the paper is to propose a solution to this problem. The solution is based on the observations that: (1) UML is a universal specification and modeling language to present data models, software architecture or business models; moreover it is supported by many tools (CASE, IDE), (2) graphical notations are easier to understand especially if the model is complex, with many relationships among model elements.

The rest of the paper is structured as follows. Section 2 presents related works and clearly states the paper's contribution. The proposed SUMO meta-model which supports the transformation process is described in Section 3. The tool and its main functional components are presented in Section 4. Newly introduced transformation rules for SUMO attributes and their relations are the subject of Section 5. Section 6 shortly defines existing transformation rules. An example of a transformation with a short discussion of its shortcomings is given in Section 7. Section 8 presents the problems to be addressed in the future. Section 9, the last one, concludes the paper.

2. Related Works

The paper [15] is the first in a series considering the SUMO ontology as a source for domain modelling. It presents an initial set of mapping rules between SUMO notions and UML notions, and identifies the elements difficult to extract, e.g. attributes.

The paper [16] presents an outline of a systematic approach to the development of a domain model on the basis of selected SUMO ontologies. It involves only a few steps. It starts with needs description, next it goes through the identification of business processes in the area of interests which help to decide if a notion of an ontology is in the area of interests (and should be translated to UML) or not. After the analysis of the selected elements, they are translated (manually) to a UML class diagram. The approach was tested on a few examples. Some SUMO-UML mappings were also refined. The biggest problems the authors found are:

- ontology size – it contains many irrelevant (out of scope) elements,
- domain knowledge is spread over many ontologies (files),
- some facts are defined at a very general level (predicates between *Object*, *Physical*) which makes the interpretation more difficult.

In the paper [17] the refined version of the approach from [16] is presented. It also consists of only a few steps, but their definition is much more formal and close to implementation needs. The main idea of the approach is a guided selection of the SUMO extract, which will be further translated to UML. The paper also proposes some new transformation rules, e.g. transformation of unary functions. The general finding of that work is that the process of knowledge extraction must be supported by a tool. Otherwise, the process, even if the results are promising, is very time-consuming, and error prone.

The contributions of this paper are as follows:

- The meta-model of SUMO notions used within a transformation process (see Section 3).
- Definition of a tool architecture (see Section 4).
- New transformation rules for SUMO attributes and their relations (see Section 5).
- Verification and correction of transformation rules defined in [15–17]; the subset of implemented rules (including the changed ones) is presented in Section 6.

The transformation process between two models can be specified and performed in many ways. If the source and targeted models are expressed in the XML language, the transformation process can be defined as the Extensible Stylesheet

Language Transformation (XSLT) and executed by a dedicated engine (see [18] for an example). This approach suffers from low readability and maintainability, this is why the transformation between meta-models is considered more often (e.g. [19]). In this approach at first the meta-models of the source and target models are prepared or adopted, and next the transformation rules between meta-classes are defined. Transformation rules can be expressed either in operative languages, like the Atlas Transformation Language (ATL), java or declarative ones like QVT-Relations. In the paper, the approach based on meta-models is in use. The SUMO meta-model is defined by the author of that paper. The UML meta-model is freely available (eclipse.uml2 framework).

The SUMO to UML transformation rules defined in [15–17] answer the question how to map elements such as classes and their hierarchies or relations and their hierarchies but they do not address SUMO attributes and their relationships. The problem with the SUMO attributes is that they are represented differently than attributes in the UML language. In SUMO the attribute is defined as "a quality which we cannot choose not to reify into subclasses of Objects" [8]. Because of that, attributes are assigned not to classes as in UML but to class instances. This paper fills this gap. The thorough analysis of SUMO relations between attributes is conducted here. On that basis the mapping of SUMO attributes to UML language is proposed. The mapping involves the definition of a UML profile, presented in Section 5.

The set of transformation rules defined in [15–17] was verified and extended in the meantime. The newly introduced transformation rules (including those defined for attributes), and the changed transformation rules with their justification are presented in Section 6.

3. Meta-model of SUMO Notions

To support the SUMO to UML transformation process the content of SUO-KIF files has to be represented at the higher abstraction level, which enables both: checking static consistency rules and performing the transformation process itself. It is achieved with so called meta-model of SUMO notions – see Fig. 1. The initial version of the meta-model was presented in [20]. Here the diagram is extended by new meta-classes.

The diagram reflects the logical structure of the SUO-KIF file which can be perceived as a set of sentences. A SUMO sentence is represented by the *Sentence* abstract class – the parent of all possible kinds of statements in SUMO. Each sentence belongs to exactly one *OntologySegment* (SUO-KIF file). Below there is a short description of concrete sentence classes:

1. *LogicalSentence* – a sentence starting with a logical operator, e.g. "(=> . . .), (<=> . . .)"; a tautology built with an implication and/or an equivalence operator;
2. *QuantifiedSentence* – a sentence starting either with a universal or existential quantifier: "(forall . . .) or (exists . . .)";
3. *RelationalSentence* – a sentence starting with a name of function or relation: "(name . . .)"; a fact in the considered domain stating, for example, that John likes Karin.

It is assumed that only sentences written at the first level are instantiated by the SUMO to UML translator, e.g. the text: "(=> (instance ?REL BinaryPredicate) (valence ?REL 2))" will be instantiated as one sentence even if it contains two internal sub-sentences. The parser omits SUMO comments.

The right side of the class diagram shows the structure of SUMO notions. The *Entity* is "the root node of the ontology" [8]. It is associated with all sentences it belongs to (as a part).

Entity is the parent for two UML classes interesting in the context of the considered transformation:

– *Relation* – definition of a SUMO relation or function, together with its domains and/or range (see Fig. 2);
– *Type* – represents a SUMO notion that can be instantiated, e.g. *BinaryPredicate*; types that represent SUMO *Attributes* are distinguished with *isAttribute=true* field.

Each instance of *RelationalSentence* is linked to one *Relation* (*basicRelation* role) and many

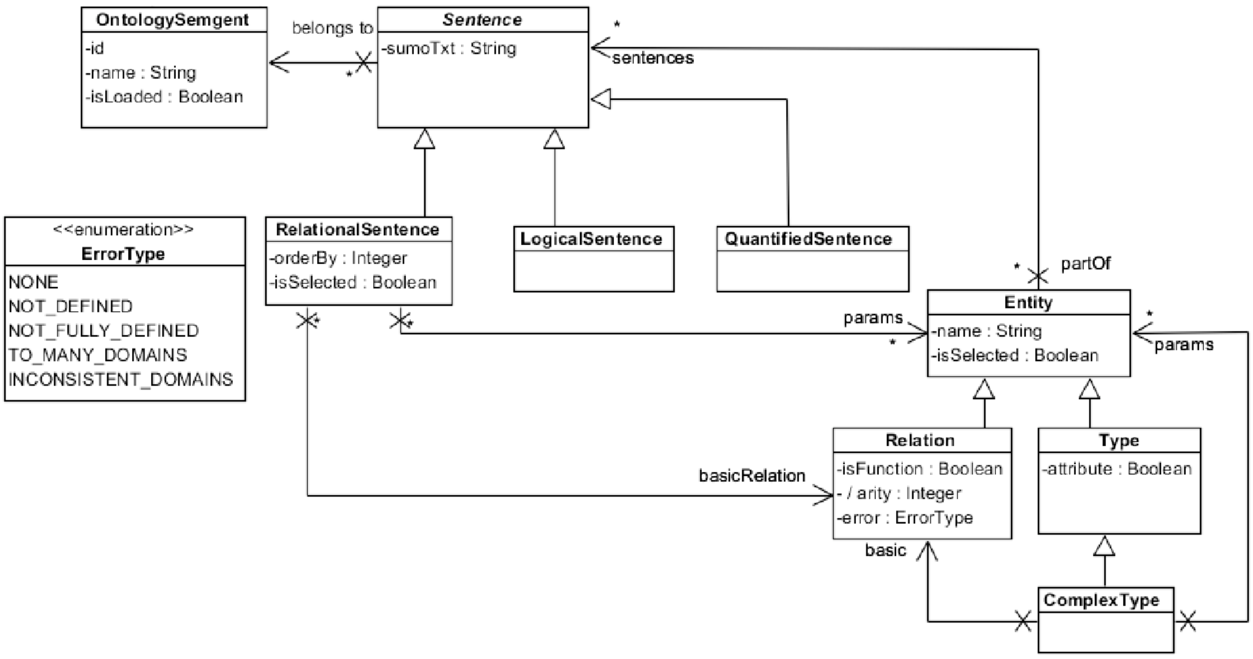


Figure 1. Meta-model of SUMO notions – main elements

Entities involved (*params* role), e.g. the sentence: "(domain part 1 Object)" is linked to *domain* relation, and has three parameters.

Sometimes relational sentences point out a type indirectly by referencing to a function which returns a type; see the sentence: "(subclass Fodder (FoodForFn DomesticAnimal))" for example. *Fodder* is a subclass of the type returned by the function *FoodForFn* called with the *DomesticAnimal* parameter. According to the specification this function returns a subclass of *SelfConnectedObject*. Such cases are represented in the proposed SUMO meta-model by a *ComplexType* class. An instance of the *ComplexType* class refers to the function it is built upon (*basic* role) – *FoodForFn* – and remembers the function parameters (*params* role) – *DomesticAnimal*.

Some specific relational sentences (defined in the SUMO upper ontology) play a crucial role in the transformation process. Up to now seven types of such sentences have been identified:

1. *Documentation* sentence (*DocumentationSent*) – a sentence starting with "(documentation...)"; contains documentation (an instance of *SymbolicString*) in a specific language for a specific entity;
2. *Instance* sentence (*InstanceSent*) – a sentence starting with "(instance...)"; is associated with an entity (instance) and a type for that instance;
3. *Subclass* sentence (*SubclassSent*) – a sentence starting with "(subclass...)"; used to describe inheritance hierarchy between SUMO classes; it is associated with parent and child types;
4. *Subrelation* sentence (*SubrelationSent*) – a sentence starting with "(subrelation...)"; allows to describe the inheritance hierarchy between SUMO relations; it is associated with parent and child relations;
5. *Domain* sentence (*DomainSent*) – a sentence starting either with "(domain...)" or "(domainSubclass...)"; it represents the domain element (*Type*) for a specific relation;
6. *Range* sentence (*RangeSent*) – a sentence starting either with "(range...)" or "(rangeSubclass...)"; represents a range (*Type*) for a function (*Relation* with *isFunction* attribute set to true);
7. *Partition* sentence (*PartitionSent*) – a sentence starting either with "(partition...)" or "(disjointDecomposition...)" or "(exhaustiveDecomposition...)"; all sentences represent partition of class *C* into subclasses but

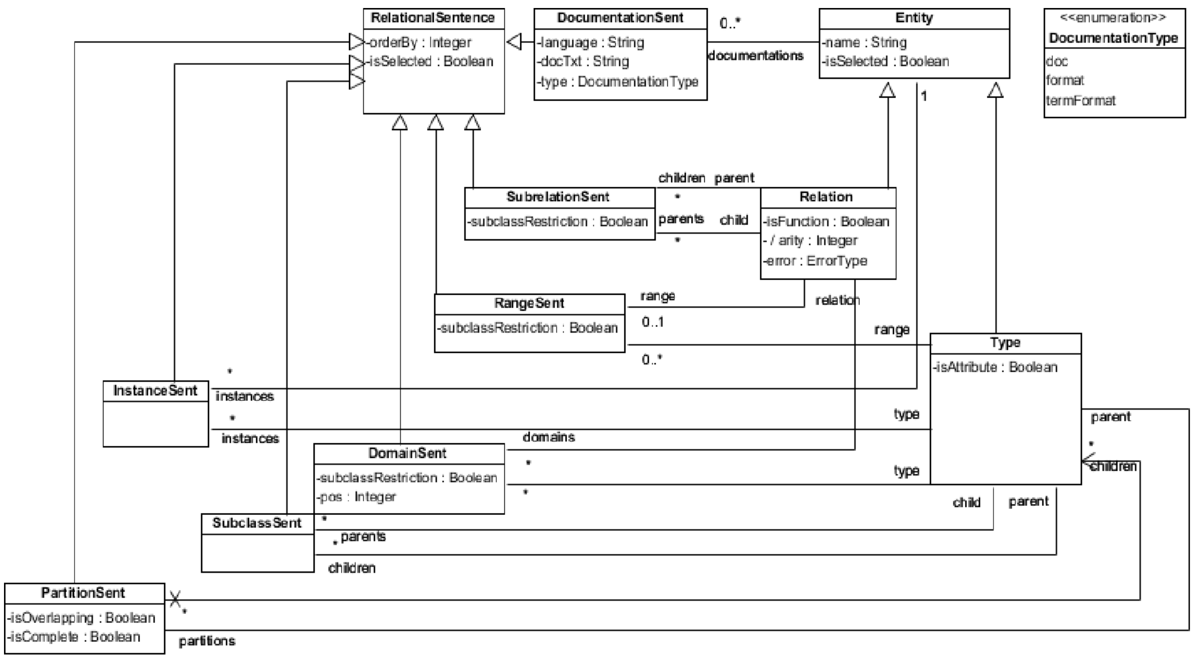


Figure 2. Meta-model of SUMO notions – hierarchy of relational sentences

they are characterized by different properties represented by *PartitionSent* attributes (*isOverlapping*, *isComplete*); i.e. a normal *partition* assumes that the subclasses are mutually disjoint and cover *C*; *disjointDecomposition* requires only that the subclasses are disjoint; and *exhaustiveDecomposition* disallows to have instances of *C* which do not belong to any of its subclasses (the subclasses do not need to be disjoint).

4. Architecture of the SUMO to UML Translator

The SUMO to UML translator is implemented in Java 8 with the Swing library. The main functional elements of the translator are presented on a component diagram – see Fig. 3.

An end-user is allowed to select any subset of ontology SUO-KIF files (called ontology segments) to be read by the tool. The loading process is controlled by a *SumoLoadConttroller* component and is presented – with the use of a sequence diagram – in Fig. 4.

SumoLoadController runs *SumoParser* to: (a) check the syntax correctness of the file, (b) walk through all tokens in the file and call *SumoModelBuilder* to translate SUMO sentences into an internal SUMO meta-model representation. *SumoParser* was generated with antlr [21] from the SUO-KIF context-free grammar [22].

Unfortunately, it turned out that SUMO ontology suffered from some bugs that could not be found by the parser (according to the rules formulated in context-free grammar). The bugs could negatively influence the correctness of the intended transformation process. So, there was a strong need to implement the *SumoChecker* component whose main functionality is to perform different consistency checks. The buggy elements are marked and reported by the tool, so the user has an opportunity to correct the input.

As it was mentioned in the previous Section, domain knowledge is spread over different SUO-KIF files which is not very convenient for transformation. That is why a separate component – *SumoReasoner* – was introduced. Its main responsibility is to update the previously generated SUMO model by inferring information indirectly defined in SUMO, e.g.: a subrelation could

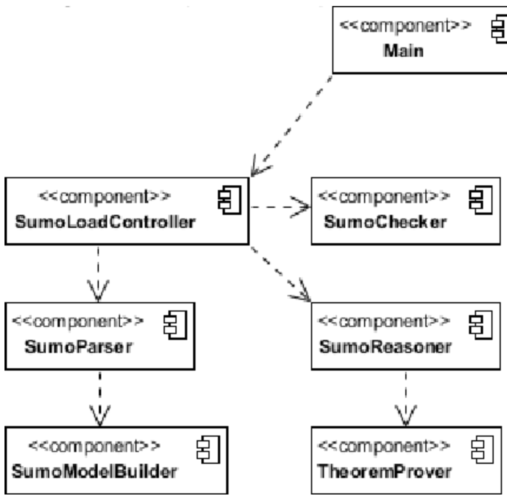

<<component>> Main
<<component>> SumoLoadController
<<component>> SumoChecker
<<component>> SumoUMLTranslator
<<component>> SumoParser
<<component>> SumoReasoner
<<component>> SumoModelBuilder
<<component>> TheoremProver

Figure 3. Architecture (functional view) of SUMO to UML translator

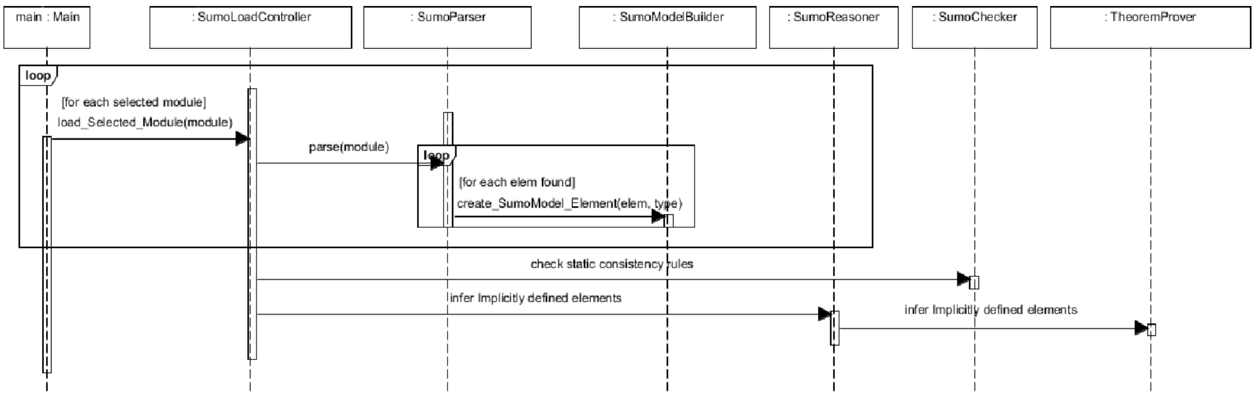

main : Main
: SumoLoadController
: SumoParser
: SumoModelBuilder
: SumoReasoner
: SumoChecker
: TheoremProver
loop
[for each selected module]
load_Selected_Module(module)
parse(module)
loop
[for each elem found]
create_SumoModel_Element(elem, type)
check static consistency rules
infer implicitly defined elements
infer implicitly defined elements

Figure 4. Processing of an ontology segment

inherit the domain definition from its parents; in such a case *SumoReasoner* copies domains from the parent to all its children.

It is also planned (this feature has not been implemented yet), that *SumoReasoner* will communicate with a selected theorem prover to reason knowledge from the rules. The new version of the Sigma tool [10] is prepared to collaborate with E prover [23]. E prover can deliver answers for specifically marked conjecture formulas. Sigma implemented mapping rules between SUO-KIF and TPTP formalism used by E prover. In consequence, a user can formulate questions like: "(instance ?X BinaryPredicate)" to find out all instances of *BinaryPredicate*.

The *SumoUMLTranslator* component conducts the transformation process. It produces – with the use of eclipse.emf and eclipse.uml2 frameworks – an instance of a UML model (version 2.5 [24]) and stores it in a file (*.uml) which can be read in a form of a tree or can be visualized on a diagram with additional tools, like e.g. Papyrus [25].

5. Translation of SUMO Attributes and Their Relations to UML

This section presents a proposal of SUMO attributes translation to UML.

5.1. Attributes and Attributes' Relations in SUMO

The *Attribute* in SUMO is a subclass of the *Abstract* class. Instances of the *Abstract* class "cannot exist at a particular place and time without some physical encoding or embodiment" [8]. In other words, attributes represent some properties or the characteristics of instances.

The *Attribute* class has two direct subclasses (*InternalAttribute*, and *RelationalAttribute*) which in turn have many own subclasses. The hierarchy of attributes is more than five levels deep.

Attribute as a class is the domain of several SUMO relations (given below in an alphabetical order):

– *contraryAttribute*: *Attribute* x . . . x *Attribute* – is used to define "a set of Attributes such that something cannot simultaneously have more than one of these Attributes. For example, (contraryAttribute Pliable Rigid) means that nothing can be both Pliable and Rigid" [8];
– *exhaustiveAttribute*: *AttributeSubclass* x *Attribute* x . . . x *Attrbute* – "relates a class to a set of Attributes, and it means that the elements of this set exhaust the instances of the class. For example, (exhaustiveAttribute PhysicalState Solid Fluid Liquid Gas Plasma) means that there are only five instances of the class PhysicalState" [8];
– *subAttribute*: *Attribute* x *Attribute* – means that "the second argument can be ascribed to everything which has the first argument ascribed to it" [8]; it is a partial ordering relation which means that the hierarchy of attributes can form a tree;
– *successorAttribute*: *Attribute* x *Attribute* – means that the second attribute comes immediately after the first attribute on the scale that they share, e.g. "(successorAttribute DeluxeRoom SuiteRoom)"; *subAttribute* tuples have nothing in common (are disjoint) with *successorAttribute* tuples; moreover, *successorAttribute* is not a partial ordering relation which means that the involved attributes must be directly ordered;
– *successorAttributeClosure*: *Attribute* x *Attribute* – means that there is a chain of *successorAttribute* assertions connecting the first and the second parameter, e.g. "(successorAttributeClosure StandardRoom SuiteRoom)".

An assignment of an attribute instance to an entity instance can be done with a *property* relation (or one of its subrelations), e.g. "(property ?Entity ?Attr)" means that ?Entity has the attribute ?Attr.

The extended version of SUMO meta-model, covering the newly introduced relations, is presented in Fig. 5. The *successorAttributeClosure* relation is not included as it will not be translated to the UML. The meta-class representing *contraryAttribute* relation (*contraryAttributeSent*) inherits all necessary assotiations from its parent.

5.2. Mappings of SUMO Attributes and Attributes' Relations to UML

5.2.1. Mappings of SUMO Attributes

Transformations of SUMO notions to UML should preserve the original semantics as much as it is possible. An existing transformation rule maps any SUMO class to a UML class with the same name. This rule needs to be refined for attributes (understood as classes). As an attribute can have many instances (e.g. *Solid*, *Fluid*, *Liquid*, *Gas*, *Plasma* are instances of *PhysicalStateemph* attribute), it would be valuable to represent directly these instances on a UML class diagram. So this is why the *Attribute* class and their subclasses are mapped to a UML enumeration data type with the same name. "As a specialization of classifier, enumeration can participate in generalization relationships" [8]. This feature enables to represent also the inheritance hierarchy between *Attribute* subclasses. An enumeration value corresponds to one of user-defined enumeration literals.

These literals are used to represent attribute instances.

Not all relations between SUMO attributes can be represented graphically on a class diagram. Fortunately, UML is a very flexible language which can be extended for a specific purpose with the use of profiles.

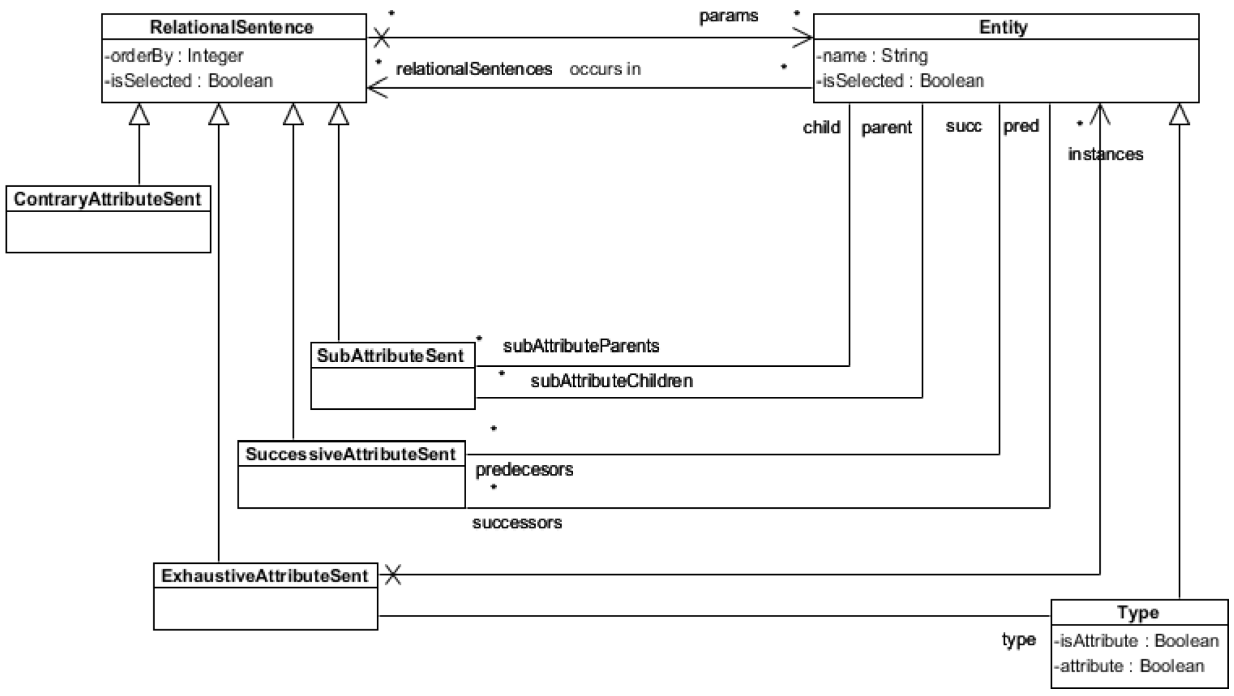


Figure 5. Extended version of the SUMO meta-model – definition of attributes' relations

5.2.2. UML Profile for Modelling SUMO Attributes

A UML profile is a lightweight extension mechanism to the UML by defining custom stereotypes, tagged values, and constraints. Profiles allow to adapt the UML metamodel for different domains [26]. UML profiles were defined for other ontology languages, e.g. OWL [27]. In the paper "UML Profile for OWL" authors define two-way mappings between the ontology definition meta-model (ODM) and the ontology UML profile.

The UML profile is defined as a specific package, containing stereotypes and constrains. These stereotypes can have meta-attributes called tagged values. "A stereotype is a profile class which defines how an existing metaclass may be extended as part of a profile. It enables the use of a platform or domain specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass" [27].

UML profile for SUMO attributes introduces only two stereotypes (see Fig. 6):

- «Attribute» which is applied to enumerations, and
- «AttributeInstance» which is applied to enumeration literals being owned by the enumeration with «Attribute» stereotype; this stereotype has one property (pos: Integer), which introduces a tag definition; its semantics is explained in subsection 5.1.

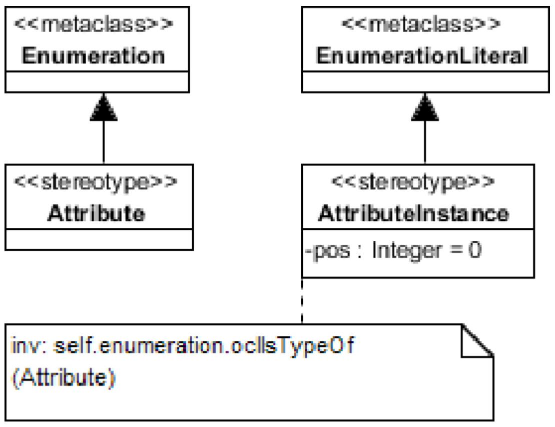


Figure 6. UML profile to represent SUMO attributes

5.2.3. Mappings of Attributes' Relations

This subchapter defines possible mappings for all relations between SUMO attributes, identified in Section 5.1, to UML language.

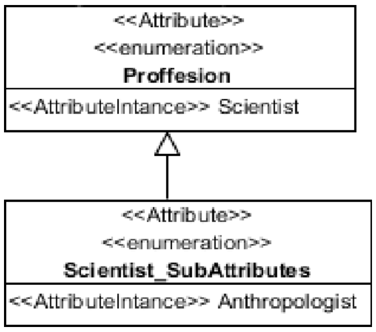
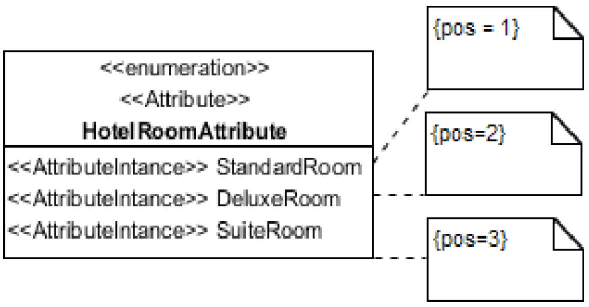


Figure 7. Transformation of the *subAttribute* relation



<<enumeration>>
<<Attribute>>
HotelRoomAttribute
<<AttributeIntance>> StandardRoom
<<AttributeIntance>> DeluxeRoom
<<AttributeIntance>> SuiteRoom
{pos = 1}
{pos=2}
{pos=3}

Figure 8. Transformation of *successorAttribute* relation

Transformation of *contraryAttribute* relation

The *contraryAttribute* relation is used to describe the fact that two specific attributes cannot be assigned to the same instance. Such a demand can be represented by an Object Constraint Language (OCL) invariant. OCL [28] is the language which enables to formally define constraints on UML models. Thus, any SUMO sentence of the form "(contraryAttribute atr1 atr2)" will be transformed as an invariant defined in the context of *Entity* class, according to the schema:

```
context Entity:
inv: not Entity.allInstances()->exists(e |
         e.hasProperty('atr1')
         and e.hasProperty('atr2'))
```

where *hasProperty(name: String): Boolean* is an auxiliary function which checks whether a specific entity e has assigned the attribute with a name equal to the input parameter.

Transformation of the *exhaustiveAttribute* relation

The *exhaustiveAttribute* relation lists all instances of a given attribute class. The list of instances cannot be further extended. To achieve the same semantics in the UML language, the UML class representing a SUMO attribute will be marked as a leaf class (*isLeaf* = *true*).

Transformation of *subAttribute* relation

The *subAttribute* relation defines the hierarchy of attribute instances. One attribute instance can be a parent for many sub-attributes, e.g. "(subAttribute Antropologist Scientist)", "(subAttribute Archeologist Scientist)". It would be valuable to present all these sub-attributes directly on a UML class diagram in the same way the other attributes' instances are represented, i.e. as enumeration literals. However, the children of a specific instance should be grouped together.

To achieve the demands mentioned above the following transformation rule is proposed. Each sentence of the form "(subAttribute atrSpec atrGen)" will be transformed according to the schema:

– If it does not exists a new artificial enumeration data type with «Attribute» stereotype and name *atrGen_SubAttributes* is created, e.g. *Scientists_SubAttributes*; the newly created enumeration will inherit from the enumeration data type for which *atrGen* is an enumeration literal; in the example the *Scientists_SubAttributes* enumeration data type will inherit from the *Proffesion* enumeration data type (see Fig. 7);
– *atrSpec* is defined as a new enumeration literal in the *atrGen_SubAttributes* enumeration data type; e.g. the *Anthropologist* enumeration literal will be added to the *Scientists_SubAttributes* enumeration.

Transformation of the *successorAttribute* relation

The *successorAttribute* relation defines the direct order between attributes. Such an order can be represented by UML tag definitions ({pos = value}). An attribute instance which is the first "in the queue" will have pos set to 1, its direct successor – pos set to 2, etc. For example, see Fig. 8 on which the transformation of SUMO sentence: "(successorAttribute StandardRoom DeluxeRoom)" is presented.

Transformation of the *successorAttributeClosure* relation

The *successorAttributeClosure* relation can be inferred from *successorAttribute* relation, and this is why it is not translated to UML.

6. Examples of Transformation Rules

This section shortly presents the implemented transformation rules focusing on those that were changed in comparison to the previous publications [15–17]. Selected transformation rules are described below.

6.1. Rule 1

SUMO Element: Direct or indirect subclass of *Entity*, e.g. *City*, *Nation*
UML Element: Class
Comment: Data values like *Integers* are also represented as separate classes (which results in uniform representation of relations).

6.2. Rule 2

SUMO Element: Binary (including self) and higher arity relations with all domains defined in the form "(domain *relation int class*)", e.g. "(domain citizen 1 Human)", "(domain citizen 2 Nation)"
UML Element: Association with a proper arity, e.g. *citizen*, *capitalCity*
Comment: Previously, when one of the domains in a relation was a data value, e.g. *Integer*, the relation was represented either as an attribute (for a binary relation) or an association class; now, all binary or higher arity relations are represented in the same way as associations.

6.3. Rule 3

SUMO Element: A relation domain or a function range defined in the form "(domainSubclass *relation int class*)", e.g. "(domainSubclass roomAmenity 1 HotelUnit)", or "(rangeSublcass *function class*)", e.g. "(rangeSublcass FoodForFn SelfConnectedObject)"
UML Element: Generalization set, e.g. *HotelUnit_ - Subclasses*, *SelfConnectedObject_ Subclasses*
Comment: *domainSublcass* is a constraint meaning that the int'th element of each tuple in relation must be a subclass of a specific class; similarly, *rangeSubclass* stays the same for function ranges; that this notion is represented by the UML generalization set.

6.4. Rule 4

SUMO Element: Binary (including self) and higher arity relations for which at least one domain is defined in the form "(domainSublcass *relation int class*)", e.g. "(domainSublcass roomAmenity 1 HotelUnit)", "(domainSubclass roomAmenity 2 Physical)"
UML Element: Association among the results of the transformation of relation domains including generalization sets, e.g. *roomAmenity* (association between *Physical_ Subclasses* and *HotelUnit_ Subclasses*)
Comment: The previous transformation was incorrect (misinterpreted semantics); the association used to link classes; the new association links generalization sets.

6.5. Rule 5

SUMO Element: Subrelation relationship "(subrelation *child-relation parent-relation*)" e.g."(subrelation geographicSubregion located)"
UML Element: An association with a "subsetted" property; the association ends of the *child-relation* will be the subsets of association ends of *parent-relation*; e.g. *geographicSubregion* association ends will be the subsets of *located* association ends
Comment: A *subrelation* is a constraint meaning that every tuple of a child relation is also a tuple of a parent relation; in the UML 2.5 such a feature is represented by a subset constraint.

6.6. Rule 6

SUMO Element: Partition relationship in the form "(partition *C C1 C2*...)"
UML Element: Generalization set with *isOver-*

lapping=false and *isComplete=true*
Comment: New.

6.7. Rule 7

SUMO Element: Exhaustive decomposition relationship in the form "(exhaustiveDecomposition *C C1 C2*...)"
UML Element: Generalization set with *isOverlapping=true* and *isComplete=false*
Comment: New.

6.8. Rule 8

SUMO Element: Disjoint decomposition relationship in the form "(disjointDecomposition *C C1 C2*...)"
UML Element: Generalization set with *isOverlapping=false* and *isComplete=false*
Comment: New.

6.9. Rule 9

SUMO Element: *Attribute* class or its subclass
UML Element: Enumeration data type with «Attribute» stereotype
Comment: New.

6.10. Rule 10

SUMO Element: *Attribute* instance
UML Element: Enumeration literal with «AttributeInstance» stereotype in the enumeration data type
Comment: New.

6.11. Rule 11

SUMO Element: subclass relation between *Attribute* classes, e.g. "(subclass HotelRoomAttribute RelationalAttribute)"
UML Element: Generalization relationship between enumerations
Comment: New.

6.12. Rule 12

SUMO Element: *contraryAttribute* relation, e.g. "(contraryAttribute Dirty Clean)"
UML Element: An OCL invariant defined in the context of *Entity* class
Comment: New.

6.13. Rule 13

SUMO Element: *exhaustiveAttribute* relation, e.g. "(exhaustiveAttribute SexAttribute Female Male)"
UML Element: Property *isLeaf* in the class representing the attribute is set to true
Comment: New.

6.14. Rule 14

SUMO Element: *subAttribute* relation, e.g. "(subAttribute Anthropologist Scientist)"
UML Element: A new enumeration data type gathering all sub attributes (left parameter) of the right parameter as literals; this new data type inherits from the enumeration data type representing the right parameter
Comment: New.

6.15. Rule 15

SUMO Element: *successorAttribute* relation, e.g. "(successorAttribute StandardRoom DeluxeRoom)"
UML Element: Tag definitions assigned to enumeration literals with pos tag set to the order number of the attribute instance
Comment: New.

7. SUMO to UML Transformation Example

The functionality of SUMO to UML translator will be presented with the use of a simple example. It aims at elaborating an initial version of domain diagram based on the *Countries and Regions* ontology and the ontologies it is based upon (e.g. Merge.kif, Mid-level-ontology.kif, Goverment.kif, all downloaded on the 1st January 2016) [8]. Figure 9 shows a form which allows a user to select interesting ontologies (ontology segments).

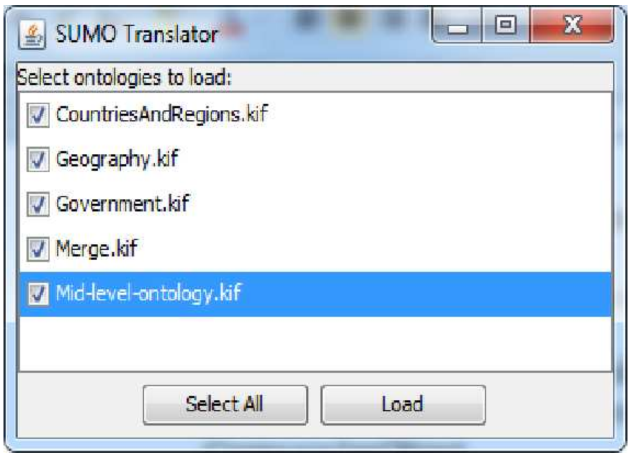


Figure 9. SUMO to UML translator – the initial form

After file loading the *SumoChecker* component reports found bugs. SUMO sentences which are the source of bugs are marked in red in the main window. Examples of such bugs are presented below:

– Entity: *DeviceNormal* has two different informal documentation sets.
– Relation: *defendant* 1st domain: *CognitiveAgent* does not fit parent: *patient* domain: *Process*.
– Type: *PostalAddressText* lacks its documentation.

Let us assume that a user wants to propose a domain model to represent the structure of geographic areas, their types, inclusions, as well as capital cities for particular geopolitical regions. He needs to find among SUMO notions those to be translated to UML and to mark them. The tool helps to identify interesting concepts by providing all sentences in which a given concept is used, grouped by their type; for example, for a relation the documentation sentence is presented first, next relation domains, sub-relations and relation instances (see Fig. 10).

Within the main window, a user can search or browse SUMO content. On the left there is a list of all entities found in selected SUMO ontologies. Because the number of entities is huge, the view could be limited only to entities whose names start with a specific letter. On the right, there is a set of sentences the entity is part of. There is also Rule tab containing axioms referring to a selected entity.

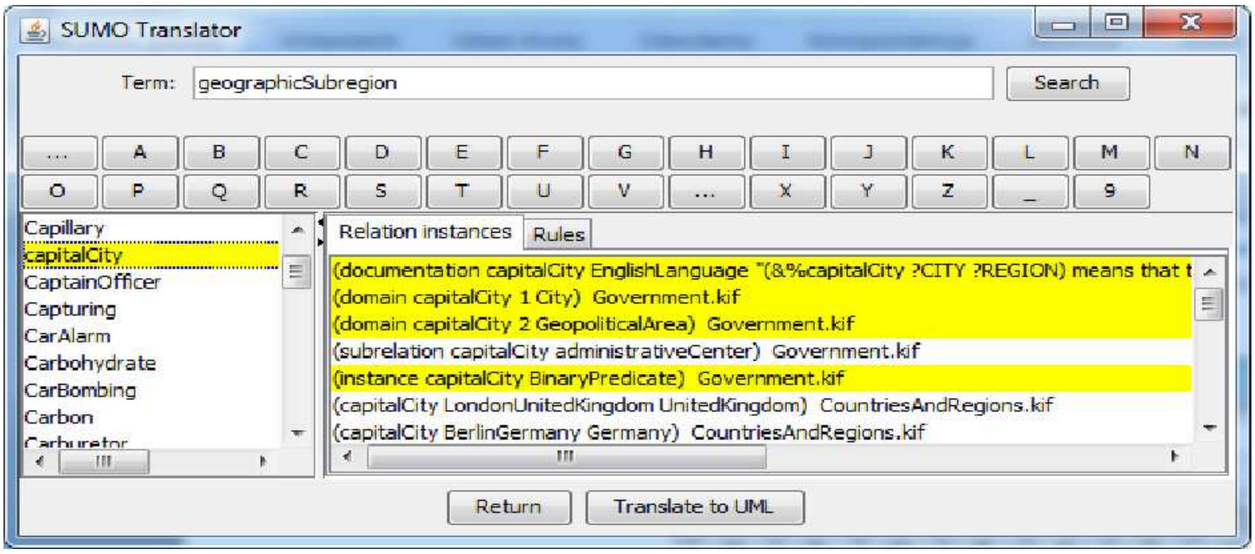
By a double click a user can select either entities or sentences to be translated to UML. Selected elements are marked in yellow – see Fig. 10. If a relation is selected, its domains are automatically selected as well. For example, among relationships in which the *City* class is involved, *capitalCity* was chosen to be translated into UML. When the selection process is completed, the user runs the translation process.

Figure 11 presents the result of a transformation made by the translator. The resulting UML class diagram has a form of a tree with properties set for classes and associations.

For readability purposes the generated file was rewritten in the Visual Paradigm tool and presented as a graph in Fig. 12. Examples of elements that cannot be visualized (e.g. *subset* constraint for association ends) are given in comments.

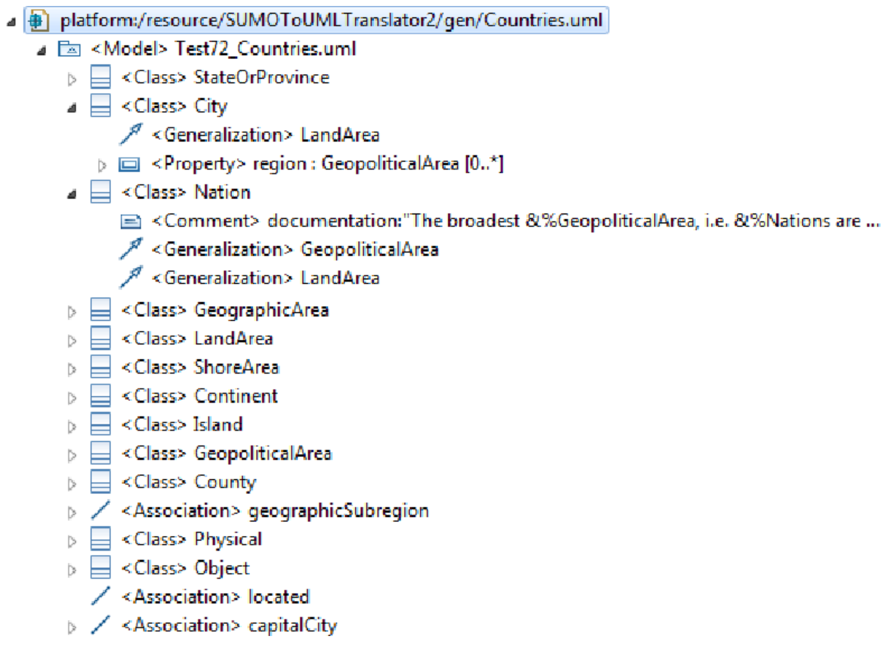
As one can observe, the resulting class diagram may consist of more than one sub-graphs – see the *located* association between *Object* and *Physical* classes. There could be the following reasons for that:

– The user did not mark SUMO sentences describing the inheritance hierarchy to be transformed; e.g. *GeographicArea* is an indirect child of *Object* and *Physical* which means, that – in this context – *located* relation can happen between *GeographicAreas*.
– Some knowledge is contained in qualified sentences which are not processed at that moment in any way.



(documentation capitalCity EnglishLanguage "(&%capitalCity ?CITY ?REGION) means that t
(domain capitalCity 1 City) Government.kif
(domain capitalCity 2 GeopoliticalArea) Government.kif
(subrelation capitalCity administrativeCenter) Government.kif
(instance capitalCity BinaryPredicate) Government.kif
(capitalCity LondonUnitedKingdom UnitedKingdom) CountriesAndRegions.kif
(capitalCity BerlinGermany Germany) CountriesAndRegions.kif

Figure 10. SUMO to UML translator – the main window



```
platform:/resource/SUMOToUMLTranslator2/gen/Countries.uml
  <Model> Test72_Countries.uml
    <Class> StateOrProvince
    <Class> City
      <Generalization> LandArea
      <Property> region : GeopoliticalArea [0..*]
    <Class> Nation
      <Comment> documentation:"The broadest &%GeopoliticalArea, i.e. &%Nations are ...
      <Generalization> GeopoliticalArea
      <Generalization> LandArea
    <Class> GeographicArea
    <Class> LandArea
    <Class> ShoreArea
    <Class> Continent
    <Class> Island
    <Class> GeopoliticalArea
    <Class> County
    <Association> geographicSubregion
    <Class> Physical
    <Class> Object
    <Association> located
    <Association> capitalCity
```

Figure 11. SUMO to UML transformation example (automatic translation)

SUMO ontologies form a set. The upper layer is included in Merge.kif file. At this level, many basic relations are defined, including *located*, so this is why their domains are top classes from SUMO class hierarchy (*Physical* and *Object* for *located* relation). When considering a specific domain, e.g. countries and regions, one deals with subclasses of the top level classes; the instances of these subclasses can be used in all places where their parents are allowed. It means that an interesting relation could be defined between classes being far away (in the inheritance hierarchy) from classes of the considered domain. To solve this problem, the translation tool can add indirect inheritance relationships between classes presented on the class diagram.

The domain diagram resulting from the transformation process is a starting point to understand a given domain. It is consistent with domain ontology by construction, but it can lack

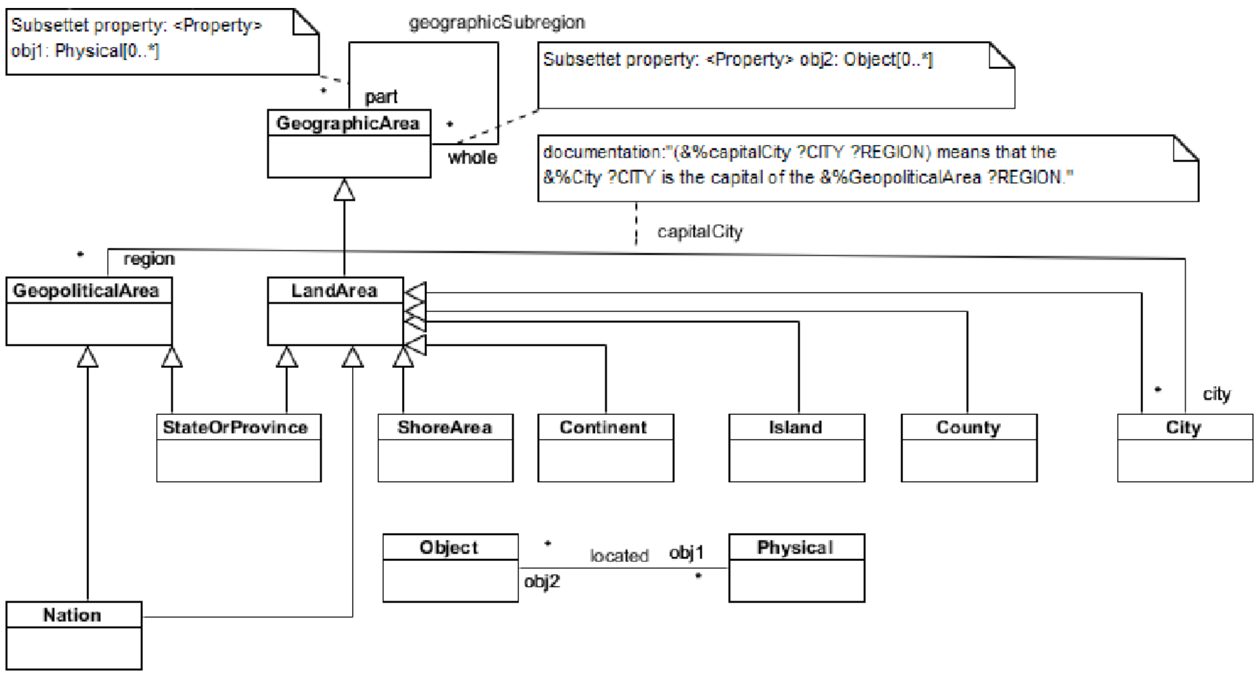

Subsettet property: <Property>
obj1: Physical[0..*]
geographicSubregion
Subsettet property: <Property> obj2: Object[0..*]
part
GeographicArea
whole
documentation:"(&%capitalCity ?CITY ?REGION) means that the &%City ?CITY is the capital of the &%GeopoliticalArea ?REGION."
capitalCity
region
GeopoliticalArea
LandArea
StateOrProvince
ShoreArea
Continent
Island
County
city
City
Object
located
obj1
obj2
Physical
Nation

Figure 12. SUMO to UML transformation example – results presented as a class diagram

some important information. The quality of the diagram strongly depends on the initial step performed by a system analyst – identification of SUMO notions to be translated. This problem is addressed in [29].

8. Problems to be Addressed

8.1. Meta-classes and Meta-relations

SUMO, similarly to UML, is described in SUMO itself. Some elements of SUMO play the role of meta-classes, i.e. classes the instances of which are functions or relations; examples include *BinaryPredicate*, *IrreflexiveRelation*. Meta-classes are not directly translated to a UML class diagram, but they define important properties of other transformed elements, e.g. the arity of relations or functions. At this moment, only arity is transformed. Another relation properties, e.g. the "reflexivity" constraint is not translated, but that could be done with the use of OCL.

Meta-relations are the relations describing relationships between 2 or more classes or 2 or more relations; examples include: *subclass*, *partition*, *disjoint* for classes, and *subrelation*, *disjointRelation* for relations. In the current version of the tool most of them are addressed (see Sections 4–6 for details) but still some other can be taken into consideration, e.g. *disjointRelation*.

8.2. Axioms

SUMO axioms introduce constraints on ontology instances. The example below states that every instance of *EuropeanCity* must be part of *Europe*.

```
 (=>
(instance ?CITY EuropeanCity)
(part ?CITY Europe)
)
```

The other example stays that if an instance belongs to *VirginIslands* it must be also an instance of *Island*.

```
 (=>
(member ?ISLAND VirginIslands)
(instance ?ISLAND Island)
)
```

Some of such axioms could be expressed directly in UML (e.g. with the use of a composition relationship), some other could be translated into OCL. The current version of the SUMO to UML transformation tool allows reading axioms but they cannot be selected for transformation.

9. Summary

The paper presents an approach to SUMO-UML translation. The translation is defined as a set of transformation rules between SUMO and UML meta-models.

The SUMO meta-model was proposed for this purpose. The initial set of transformation rules, identified and described in [15–17], was revised and extended with new rules e.g. for SUMO attributes and their relations.

The results of the tool applications are promising. The obtained domain class diagrams are consistent, correct and complete to the level to which the input ontology has these features. These are the main benefits the tool can bring to potential users. Business expert or business analyst can use the tool to find out interesting notions, select them, and translate to a UML class diagram with a set of OCL constraints with one click. The user is warned about incompleteness and inconsistencies found in the original files. He or she can experiment with transformation results, selecting new elements or un-selecting the previously selected ones. The obtained UML model can be re-factored, and next transformed to other representations, e.g. programming languages, database schemas, etc.

The tool to be effectively used needs a qualified business analyst or business expert to select all interesting SUMO notions for transformation. Otherwise, the resulting domain model will be incomplete. To address this matter a research group, the author of this paper belongs to, is trying to propose an algorithm to extract knowledge from ontology on the basis of limited input only – see [29].

A kind of a side effect of the tool implementation is the definition of static consistency rules which allow to detect inconsistencies in existing ontologies. In the future, this module can be used as a part of an ontology editor.

The next release of the tool will address problems presented in Section 8. Additionally, the transformations at the instance level, represented by object diagrams, are planned to be implemented. It seems to be especially important because in domain ontologies more than half of sentences represent instances and links among them, e.g. "(instance Mauritius Nation) (geographicSubregion Mayotte SouthernAfrica)" for CountiresAndRegions.kif.

References

[1] K. Bittner and I. Spencer, *Use Case Modeling.* Addison-Wesley Professional, 2002.

[2] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software development process.* Addison-Wesley Professional, 1999.

[3] D. Zowghi and V. Gervasi, "The three cs of requirements: Consistency, completeness, and correctness," in *Proceedings of 8th International Workshop on Requirements Engineering: Foundation for Software Quality, (REFSQ '02*, 2002, pp. 155–164.

[4] P. Mohagheghi, V. Dehlen, and T. Neple, "Definitions and approaches to model quality in model-based software development – a review of literature," *Information and Software Technology*, Vol. 51, No. 12, Dec. 2009, pp. 1646–1669.

[5] W. Hesse, "Ontologies in the software engineering process," in *EAI 2005: Enterprise Application Integration – Proceedings of the 2nd GI-Workshop on Enterprise Application Integration*, R. Lenz, U. Hasenkamp, W. Hasselbring, and M. Reichert, Eds., 2005.

[6] H.J. Happel and S. Seedorf, "Applications of ontologies in software engineering," in *Proc. of Workshop on Sematic Web Enabled Software Engineering"(SWESE) on the ISWC*, 2006, pp. 5–9.

[7] F. Gailly and G. Poels, "Conceptual modeling using domain ontologies: Improving the domain- specific quality of conceptual schemas," in *Proceedings of the 10th Workshop on Domain-Specific Modeling*, ser. DSM '10. New York, NY, USA: ACM, 2010, pp. 18:1–18:6.

[8] Suggested Upper Merged Ontology, last access: 10 Jan 2016. [Online]. http://www.ontologyportal.org

[9] A. Pease, *Ontology: A practical Guide.* Articulate Software Press, 2011.

[10] Sigma, last access: 10 Jan 2016. [Online]. http://sourceforge.net/projects/sigmakee/files/

[11] I. Istochnick, OWL2UML, last access: 10 Jan 2016. [Online]. http://protegewiki.stanford.edu/wiki/OWL2UML

[12] F. Suchanek, “Ontological reasoning for natural language understanding,” Master Thesis in Computer Science, Saarland University, Germany, March 2005.

[13] Semantics of Business Vocabulary and Business Rules (SBVR). Version 1.3, OMG, (2015, May). [Online]. http://www.omg.org/spec/SBVR/1.3/

[14] A. Marinos, S. Moschoyiannis, and P.J. Krause, “An SBVR to SQL compiler,” in *Proceedings of the RuleML-2010 Challenge, at the 4th International Web Rule Symposium*, 2010.

[15] B. Hnatkowska, Z. Huzar, I. Dubielewicz, and L. Tuzinkiewicz, “Problems of SUMO-like ontology usage in domain modelling,” in *Intelligent Information and Database Systems*, ser. Lecture Notes in Computer Science, N. Nguyen, B. Attachoo, B. Trawinski, and K. Somboonviwat, Eds. Springer International Publishing, 2014, Vol. 8397, pp. 352–363.

[16] I. Dubielewicz, B. Hnatkowska, Z. Huzar, and L. Tuzinkiewicz, “Domain modelling in the context of ontology,” *Foundations of Computing and Decision Sciences*, Vol. Volume 40, No. 1, 2015, pp. 3–15.

[17] B. Hnatkowska, Z. Huzar, I. Dubielewicz, and L. Tuzinkiewicz, “Development of domain model based on SUMO ontology,” in *Theory and Engineering of Complex Systems and Dependability*, ser. Advances in Intelligent Systems and Computing, W. Zamojski, J. Mazurkiewicz, J. Sugier, T. Walkowiak, and J. Kacprzyk, Eds. Springer International Publishing, 2015, Vol. 365, pp. 163–173.

[18] D. Gasevic, D. Djuric, V. Devedzic, and V. Damjanovi, “Converting UML to OWL ontologies,” in *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers &Amp; Posters*, ser. WWW Alt. ’04. New York, NY, USA: ACM, 2004, pp. 488–489.

[19] J. Zedlitz, J. Jörke, and N. Luttenberger, *Knowledge Technology*. Berlin, Heidelberg: Springer-Verlag, 2012, ch. From UML to OWL 2, pp. 154–163.

[20] B. Hnatkowska, *From requirements to software: research and practice*. Warszawa: Polish Information Processing Society, 2015, ch. Towards automatic Sumo to UML translation, pp. 87–99.

[21] ANTLR, last access: 10 Jan 2016. [Online]. http://www.antlr.org/

[22] A. Pease, Standard upper ontology knowledge interchange format, (2009). [Online]. http://sigmakee.cvs.sourceforge.net/viewvc/sigmakee/sigma/suo-kif.pdf

[23] S. Schulz, “System description: E 1.8,” in *Logic for Programming, Artificial Intelligence, and Reasoning*, ser. Lecture Notes in Computer Science, K. McMillan, A. Middeldorp, and A. Voronkov, Eds. Berlin Heidelberg: Springer-Verlag, 2013, Vol. 8312, pp. 735–743.

[24] Unified Modeling Language. Version 2.5, OMG, (2013, September). [Online]. http://www.omg.org/spec/UML/

[25] Papyrus modeling environment, last access: 10 Jan 2016. [Online]. http://www.eclipse.org/papyrus/

[26] UML profile diagrams, last access: 28 May 2016. [Online]. http://www.uml-diagrams.org/profile-diagrams.html

[27] D. Djurić, D. Gašević, V. Devedžic, and V. Damjanović, *Proceedings of the Web Engineering: 4th International Conference*. Berlin, Heidelberg: Springer-Verlag, 2004, ch. UML Profile for OWL, pp. 607–608.

[28] Object Constraint Language. Version 2.4, OMG, (2014, February). [Online]. http://www.omg.org/spec/OCL/2.4/

[29] B. Hnatkowska, Z. Huzar, L. Tuzinkiewicz, and I. Dubielewicz, *Intelligent Information and Database Systems*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 2016, Vol. 6592, ch. Conceptual Modeling Using Knowledge of Domain Ontology, pp. 554–564.