FOUNDATIONS OF COMPUTING AND DECISION SCIENCES
Vol. 40 (2015) No. 1
DE GRUYTER OPEN
DOI: 10.1515/fcds-2015-0001
ISSN 0867-6356
e-ISSN 2300-3405

DOMAIN MODELING IN THE CONTEXT OF ONTOLOGY

Iwona DUBIELEWICZ, Bogumiła HNATKOWSKA, Zbigniew HUZAR,
Lech TUZINKIEWICZ*

Abstract. The domain knowledge represented by ontology should be widely used in the design process of information system. The aim of the paper is to outline a systematic approach of developing a CIM model (domain model, precisely) on the basis of a selected domain ontology. There are presented some hints how ontology concepts can be expressed in domain model. Elaborated example realizes some difficulties in proposed approach, e.g. the domain knowledge is spread over many ontologies, some facts are defined at very general level (their interpretation is more difficult), ontology may contain many irrelevant elements. Nevertheless, we are believed that applying ontology in conscious way can help to achieve higher compliance of the domain model with the application domain.

Keywords: domain modelling, ontology, MDA, CIM, UML, SUMO

1. Introduction

In the course of a couple of few last years a specific tendency in the approach to software development has been observed. Namely, the emphasis is put on the initial stages of software development process. This is justified by the observed experience, that the quality of these stages affects the quality of the final software product and the efficiency of the development process. Assuming the Model Driven Architecture (MDA) approach to software development, the establishment of high-quality Computation Independent Model (CIM) becomes a major problem. The motivation of the paper is to analyze how the usage of domain ontologies may facilitate elaboration of CIM models.

The aim of the paper is to outline a way how to develop a CIM model on the basis of a given domain ontology, and a set of needs for a software system. CIM models, similarly to other kind of models, should represent both static and dynamic aspect of the domain of interest. Considerations presented in the paper are restricted to the static aspect only.

Ontologies have become popular in several fields of information technologies like software engineering and database design. The domain knowledge represented by ontology should be widely used in the design process of information systems. For example, in [1]

* Wrocław University of Technology, Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland, Bogumila.Hnatkowska@pwr.edu.pl

and [2] authors analyze the role of ontologies in software engineering processes. They believe that ontologies are an important source of knowledge in the conceptualization phase and propose the ontology life cycle as the background for software development. Similar view in presented in [3] with suggestion that software modeling can be improved by the integration with ontology. The paper [4] shows that ontologies can be a basis for domain analysis and simulation in software development. A framework that supports the software development process based on ontologies is presented in [5]. An application of domain ontologies to conceptual model development is presented [6].

Special role of ontologies is perceived for database development [7]. A method of quality evaluation of conceptual data model based on ontology is presented [8]. An ontology use in the process of data integration is a subject of [9].

The paper is structured as follows. Section 2 explains what ontologies are, and Section 3 describes domain models. In the next Section 4, the problem of representation of SUMO ontology elements by UML-like elements in domain models is discussed. An approach to domain model defining is outlined in Section 5. Finally, Section 6 summarizes the paper with final conclusions.

2. What are ontologies?

The notion of ontology is used in different contexts. In philosophy, ontology is the study of the nature of being, the basic categories of being and their relations. In software engineering, ontology is understood as a representation of domain knowledge in the form of a set of concepts within the domain, the properties of the concepts and interrelations of those concepts. The problem of an ontology definition and representation was raised by OMG. In the Ontology Definition Metamodel [10] the notion of ontology is explained as:

Ontology defines the common terms and concepts (meaning) used to describe and represent an area of knowledge. An ontology can range in expressivity from a Taxonomy (knowledge with minimal hierarchy or a parent/child structure), to a Thesaurus (words and synonyms), to a Conceptual Model (with more complex knowledge), to a Logical Theory (with very rich, complex, consistent, and meaningful knowledge).

In general, depending on the level of abstraction three categories of ontologies are identified: upper-level (high-level), middle-level, and low-level (domain) ontologies. The categories are strongly related: a low-level ontology is developed on the basis of a middle-level, and the middle-level on a high-level ontology. Notions that are introduced and explained by ontologies request agreement upon them in the software engineering community. The main problem is which upper-level ontology is the best suited for development of lower-levels ontologies. The problem is not solved yet, and therefore IEEE raised an initiative – a series of projects, that are aimed in standardized upper level ontology elaboration. Up to now an eventual standard has not been agreed, but there are some competing proposals, for example SUMO (Suggested Upper Merged Ontology), OpenCYC [11], IFF [12].

We decided to consider SUMO as a root-ontology because it defines formally more than 1000 notions. Further 20 000 notions is defined by other middle-level and low-level (domain) ontologies that have grown up on the basis of SUMO. In SUMO the following common components may be discriminated:

1. individuals: instances or objects (the basic or "ground level" objects);
2. classes: sets, collections, concepts, types of objects, or kinds of things;
3. attributes: aspects, properties, features, characteristics, or parameters that objects (and classes) can have;
4. relations: ways in which classes and individuals can be related to each other;
5. function terms: complex structures formed from certain relations that can be used in place of an individual term in a statement;
6. axioms: assertions in a logical form that together comprise the overall theory that the ontology describes in its domain of application;
7. rules: statements in the form of an "if-then" (antecedent-consequent) sentence that describe the logical inferences that can be drawn from an assertion in a particular form.

All the elements are formally defined in SUO-KIF – a declarative, comprehensive language that provides expression of arbitrary logical sentences.

3. Domain model

Business model is aimed to describe an institution or a company in the way which enables software developers to understand the environment in which a future software system will be deployed and operated. The business model delivers the rationale of how the organization creates, delivers, and captures value in economic or other contexts. The model should describe organization units, business processes and business rules concerning given organization.

The business model forms a background for the establishment of the expected future information system. The future information system usually concerns only a fragment of the organization and consequently a fragment of its business model. This fragment is considered as a CIM model within MDA approach. This model reflects a software engineering perspective. It is a view of a system from the computation independent viewpoint; it does not show details of the structure of systems; it uses a vocabulary that is familiar to the practitioners of the domain in question.

There are many languages for business modeling. The UML seems to be the most popular and the most general modeling language. Recently, BPMN and SysML become more and more popular. There are also other languages, like BPEL, that are used for specific applications.

We consider how to construct a CIM model, represented in the UML, on the base of business ontology. The business model contains knowledge on the given domain. The knowledge may be represented in different forms; we concentrate on the case when the knowledge is presented as domain ontology. We assume an ideal situation, when a given domain is fully described by one ontology. In practice, we may find more complex situations, e.g. the domain is only partially described by an ontology, or there are several ontologies that together describe partially or fully the domain.

A CIM model should represent two aspects: static and dynamic. In general, the static aspect may be defined by two elements: a class diagram and an object diagram being an instance of this class diagram. The class diagram represents all possible configurations of the modeled domain, i.e. a set of objects and links among them, while the object diagram

represents exactly one configuration – an initial configuration. The dynamic aspect of CIM model should describe transitions between these configurations, and additionally how the states within a given configuration are changing. The dynamic aspect can be described in different ways, for example by a set of state diagrams associated to each of the objects within given configuration.

We restrict our consideration to domain models that represent a static aspect of CIM models. Taking into account the components of an ontology presented above, the main components of the domain model may be similarly enumerated as:

1. A set of business classes *Cl*. Each class $C \in Cl$ represents a set of business real or abstract entities taken from the domain of interest, denoted by $Dom(C)$. The class C has assigned a set of its attributes; each attribute has a name and its data type. Each instance of this class – an object of the class – represents an individual business entity; the values of its attributes are characterization of this individual business entity. An instance of a class has its identity and is discriminated from other instances of the class.
2. A set *Ins* of selected instances from the set of classes *Cl*.
3. An instantiation relation *RIns* of the signature $RIns \subseteq Ins \times Cl$; the relation assigns each instance a class (a type of instance).
4. A partially ordered taxonomic relation *RTCl* of the signature $RTCl \subseteq Cl \times Cl$; if $<C_1,C_2> \in RTCl$ it means that the superclass C_2 is more general then the subclass C_1, i.e. $Dom(C_1) \subseteq Dom(C_2)$.
5. A family of relations *RAss* of the following signatures $Ass(C_1,\ldots,C_n) \subseteq Dom(C_1) \times \ldots \times Dom(C_n)$ for $C_1,\ldots,C_n \in Cl$, and $n \geq 2$; relations in the family are called associations; the most often used are binary associations.
6. A partially ordered taxonomic *RTAss* relation of the signature $RTAss \subseteq RAss^2$; if R_1 and R_2 are relations of the same signature, then $<R_1,R_2> \in RAss$ means that the relation R_2 is more general then the relation R_1, i.e. $R_1 \subseteq R_2$.
7. A set of constrains *Con*; the constraints are represented in the form of formulas in first-order logics; the terms of the logics are instances of classes *Cl*, classes' attributes and instances of relations *RTCl* and *RTAss*.

The structure of the CIM model follows the components listed for ontology: classes with attributes, instances, some relations. Rules and axioms are represented by constraints. Constrains relate to some aspect of business domain that is intended to assert business structure or to control or influence the behaviour of the business. They may be expressed in the Object Constraint Language (OCL) which is an inherent part of the UML standard.

4. Representation of ontology elements in domain model

The community using ontologies must agree upon them in advance. The ontologies must be sufficiently authoritative to support the investment.

Ontology is a conceptual model, and shares characteristics with more traditional data models. Hence it is worthwhile to analyze if and how ontology elements can be represented in the domain model. Below we review the list of four main ontology elements: individuals, classes, attributes and relations, given in section 2, to find their possible reflection in

domain model elements presented in section 3. At the beginning of each paragraph definitions of respective notions according to SUMO ontology are quoted.

4.1. Individuals

Object corresponds roughly to the class of ordinary objects. Examples include normal physical objects, geographical regions, and locations of processes, the complement of objects in the physical class. An object is an instance of a 'set or class' if it is included in that 'set or class'. An individual may be an instance of many classes, some of which may be subclasses of others. Thus, there is no assumption in the meaning of instance about specificity or uniqueness.

The definition of an object in SUMO is consistent with the definition of instance proposed in the domain model. In the UML instances can be represented as objects in an object diagram.

4.2. Classes

Classes differ from sets in three important respects. First, classes are not assumed to be extensional. That is, distinct classes might well have exactly the same instances. Second, classes typically have an associated condition that determines the instances of the class. So, for example, the condition human determines the class of humans. Note that some classes might satisfy their own condition (e.g., the class of abstract things is 'Abstract') and hence be instances of themselves. Third, the instances of a class may occur only once within the class, i.e. a class cannot contain duplicate instances.

All these class characteristics do not remain in contradiction to the class notion as it is used in domain model. So, an ontology class can be considered as equivalent to a class in domain model.

Subclass binary predicate can relate two *Class* elements. Subclass could be represented by generalization relationship in domain diagram. One class may have multiple superclasses and subclasses in both SUMO as well as domain model.

All SUMO classes form the complex structure, whose root is the *Entity* (the universal class of *Individuals*). This concept subsumes two partitions of entities: *Physical* and *Abstract*, where the former category includes everything that has a position in space/time, and the latter category includes everything else. The entities of *Physical* classes (and in particular *Object* subclass) seem be more appropriate from the perspective of business modeling. But on the other hand a model being *an abstract object that models certain aspect of a physical object, is a subject to abstraction and idealization* itself is a subclass of *Abstract* [13] and it consists of the abstract entities: *Classes*, *Attributes*, *Quantities*, and *Relations*. Hence both entity categories (i.e. *Physical*, and *Abstract*) are further considered.

SUMO classes may be represented by domain model classes unless they represent relationships, because in this case they will be represented by associations.

Ontology *Set* notion is assumed to be extensional i.e. two sets with the same elements are identical. *Set* can be an arbitrary stock of objects (sets have no an associated condition that determines object membership). *Sets are not assumed to be unique sets, i.e. elements of a Set may occur more than once in the set*.

SUMO *Set* could be represented by a UML predefined data type, i.e. simple data type (also primitive data type) as well as an enumeration (class with the stereotype <<enumeration>>).

SUMO *Object*, defined as a class of ordinary objects, is partitioned into two concepts: *Collection* and *SelfConnectedObject*. These both categories contain objects which consist of distinguished parts (have some structure). The object of the first category consists of disconnected (separate) parts, and relation *member* exists between them and corresponding *Collection*. The objects of the second category i.e. *SelfConnectedObject* are objects *that do not consist of two or more disconnected parts.*

The first concept seems be useful for domain modeling as it allows to define (internal) structure of modeled real objects. The *Collection* concept can be represented as an aggregation class which consists of the aggregated elements (*RAss*) and *Self-ConnectedObject* could be represented by unary association.

4.3. Attributes

Qualities which we cannot or choose not to reify into subclasses of Object; Attribute is a subclass of Abstract class.

SUMO attributes are treated as properties of entities. In the domain model attributes are defined as properties of domain classes. Any instance of a specific class has the same properties its class has.

An attributes hierarchy (e.g. *subAttribute* relation in SUMO) usually is not defined in domain model. But, if it is such a case, each attribute could be represented as a separate class which stays in generalization relationship with other appropriate classes.

Some attribute values use to be defined with *quantity* concept. To represent the quantity value in SUMO some special classes e.g. *Number*, *PhysicalQuantity* have been introduced. Such classes can be represented by UML data types, e.g. primitive types, enumerations, or complex types with own properties and operations.

4.4. Relations

There are three kinds of Relation: Predicate, Function, and List. Predicates and Functions both denote sets of ordered n-tuples. The difference between these two Classes is that Predicates cover formula-forming operators, while Functions cover term-forming operators. A List, on the other hand, is a particular ordered n-tuple.

In ontologies the relations are defined for classes. Relations in SUMO have domain defined (directly or indirectly), e.g. *partyLocated* predicate has a domain defined as: *Physical* × *Object.* So, in the UML *partlyLocated* predicate will be represented as a binary association with *partlyLocated* name linking *Physical* class and *Object*.

All SUMO relations are *InheritableRelations* so that it is possible to define their subrelations. The domain of subrelation can remain the same as for parent relation or be refined. Each subrelation will be represented as an association linking domain classes corresponding to the subrelation domain. This association will inherit from the association representing the parent of subrelation (*RTAss*).

For two objects a *part* relation is defined in SUMO. This is the basic mereological relation and definitions of all others mereological relations i.e. *properPart, piece, component, member,* use it. The *part* relation is *ReflexiveRelation* meaning that every object is a part of itself.

This relation can be represented directly in domain model by aggregation relationship. Selected subrelations of *part* relation (e.g. *component*, *piece*, *properPart*) could be represented by the composition relationship.

Disjoint with the *part* relation is *contains* relation (it is a subrelation of *partlyLocated*) which should be used for a whole with separable objects. This relation could also be represented by the aggregation relationship and, if needed, with additional OCL formulas.

Summarizing, the following representation of ontology elements in domain model is proposed (see **Błąd! Nie można odnaleźć źródła odwołania.**).

Table 1. Mapping of ontology elements to domain model elements

	Ontology elements	Domain model elements
Object diagram	Individuals, Attributes	*Ins, RIns, RAss* (if there exist association instances)
Class diagram	Classes, Attributes	*Cl*
	Relations	*RTCl*, *RAss*, *RTAss*

5. Approach to domain model definition

5.1. Introduction

The proposed approach to domain model definition based on existing ontology includes the following steps:

1. Needs description (analyst).
2. Identification of business processes (expert, analyst).
3. Identification of business rules within ontology (expert).
4. Analysis of identified business rules (analyst).
5. Definition of domain model (analyst).

The steps are performed by business analyst and/or domain expert. The first two address the problem of ontology size. An expert, to be able to point out important elements in ontology specification, needs to know what the intended use of the future software is. So, the starting point is a description of stakeholder business needs (business requirements) resulting from current problems. On that basis an expert, together with analyst, should be able to identify business processes that are connected (in a whole or partially) with the previously defined needs. Business processes use some specific artifacts and concepts that are of primary interest when developing a domain model. The responsibility of an expert is to select them (together with any related information) from ontology definition. The selected business rules are then studied by business analyst. This stage may involve analyst-expert co-operation where some explanations, extensions or comments are necessary. When analysis stage is completed, the analyst is expected to translate selected parts of ontology into notation more appropriate for software development, e.g. UML. That stage can be

partially or fully automated, dependently on the ontology language, and the establishment of transformation rules. If the transformation is fully supported by a tool it can precede the analysis stage, making it easier to business analyst (no need to know specific ontology language). Then, the definition of domain model involves some improvements or simplifications.

5.2. Example

5.2.1. Needs description

The system is going to support a hotel staff in its business activities. Internet is a typical medium of selling products nowadays, so it is crucial for any hotel manager to provide the offer that way in order to be able to compete in the market. A potential hotel client should be able to get to know hotel information including hotel rooms, room equipment, room availability and prices. The client also should be able to make a room reservation or cancel it. On the other hand, the system should support the hotel staff, particularly the hotel manager and receptionists, in their duties. Hotel manager is responsible for preparing an interesting offer which is adjusted to clients' needs, and is economically viable. The manager sets the room prices, promotions, room facilities, and room renovation periods. To do this the manager should have a relevant information, e.g. about planned guest arrivals or about history of guest stays. The reception desk is managed by trained personnel. The responsibilities of the receptionists include informing clients about hotel services, manage reservations, checking in and out of the hotel, accepting payments and preparing receipts (payments are out of scope of further considerations). Reservations' details are used to an analysis of planned hotel occupancy, schedule of rooms' renovation, and adjustment the offer to the real needs. Stays' details are used to an analysis of financial incomes and also allow adjusting the offer to the market expectation.

The hotel is characterized by the following quantities:

- 10 000 potential guests per year;
- 65 rooms of 5 types;
- two reception desks with a processing time of five minutes per client.

The following main managerial and informative problems have been identified within considered domain:

- current lack of the room reservation by Internet results in clients' dissatisfaction (and choosing another offer).
- time consuming preparation of analytical reports.

Having to improve the quality of delivered hotel services and clients satisfaction, the following needs were established:

- From the perspective of hotel clients:
  - Quick and simple reservations.
- From the perspective of hotel manager:
  - Effective management of hotel resources (rooms), and assurance of high clients' satisfaction.
  - Support for analytical reports preparation.

5.2.2. Identification of business processes

According to the needs defined in the previous section we are interested in the entities involved in the following business processes:

- Hotel and rooms information retrieving – allows to gain any interesting information about hotel services from a potential client perspective.
- Reservation management from both the hotel client and hotel staff perspective – allows to make, change, and cancel reservation.
- Guest management – allows gaining, editing and removing guest data.
- Guest servicing – includes check-in, and check-out of guests in the hotel.
- Rooms' management – allows defining details of rooms offered in the hotel (standard, equipment, availability in specific period).
- Price management – allows defining price lists for rooms.
- Statistical analysis – allows a manager to gain information about current stays, planned hotel occupancy etc.

5.2.3. Identification of business rules within ontology

Having the scope of interest limited to selected business processes, we try to identify business rules involved in them. We defined the business rules belonging to different groups. We followed the classification of business rules elaborated by von Halle [14], and partially also present in SBVR [15].

We started from terms and facts identification in Hotel ontology [16]. During the first reading we omitted all axioms (rules) concentrating only on the glossary terms and their dependencies (facts). In SUMO terms are defined as subclasses of classes from different ontologies. Facts are usually instances of predicates or subrelations. They are represented with the use of conventions:

- *fact_name*: *term1* × *term2* × … × *termN* – when domain of relation (fact) is directly defined
- *fact_name1* 'subrelation' *fact_name2* – when relation represented by *fact_name1* inherits from other relation represented by *fact_name2*

We found following terms in the scope of our interest: *HotelBuilding*, *HotelRoom*, *HotelUnit*, *HotelRoomAttribute*, *StandardRoom*, *DeluxeRoom*, *SuiteRoom*, *Honeymoon-Suite*, *HotelReservation*, *HotelRating*.

We found following interesting facts (fact represents a relation between terms).

- *roomAmenity*: *HotelUnit* × *Physical*
- *maxRoomCapacity*: *StationaryArtifact* × *Integer*
- *numberOfFloors*: *Building* × *Integer*
- *lastRenovation*: *Object* x *Year*
- *numberOccupant*: *HotelReservation* × *Integer*
- *reservedRoom*: *HotelReservation* × *HotelUnit*
- *rateDetail*: *Reservation* × *Formula*
- *numberAdultOccupant* (subrelation of *numberOccupant*)
- *numberChildOccupant* (subrelation of *numberOccupant*)

- *numberSeniorOccupant* (subrelation of *numberOccupant*)

During second reading we selected axioms (rules) somehow connected to identified terms and facts.

5.2.4. Analysis of identified business rules

During that stage first of all we investigated rules in Hotel ontology, but almost in all cases the rules were useless or too complicated to be expressed (they are based on many predicates we are not interested at all). On the other side, the rules lack some basic constraints, e.g. we couldn't find the constraint that for any reservation *reservationStart* should precede *reservationEnds*.

We noticed that some obvious relationships between classes were not represented in Hotel ontology in the way we expected. For example, the fact that a *HotelRoom* should be a part (and/or a proper part in SUMO terminology) of a *HotelBuilding* was expressed with the use of rules. Moreover, Hotel ontology lacked some facts we were interested in, so we tried to find them in other ontologies. They were:

- *postAddressText*: *PostalAddressText* x *PostalPlace* (from *Mid-level* ontology)
- *reservingEntity*: *Agent* x *Reservation* (from *Dining* ontology)
- *reservationStart*, *reservationEnds*: *TimePoint* x *Reservation* (from *Dining* ontology)
- *stays* (subrelation of inhabits: *Human* x *TemporaryResidence* from *Merge* ontology)

5.2.5. Definition of domain model

Below some basic transformation rules between particular elements of business rules into elements of UML class diagram are presented.

A term from business rules is represented as a class on the domain diagram together with their parents if necessary, i.e. if the parents have some interesting features that the class inherit (e.g. *HotelBuilding* inherits *fullName* attribute from *Entity* class).

A fact from business rules is represented either by an association (if the fact domain is formed by casual classes) or by an attribute (for binary predicates where one of domain is a simple type, e.g. Integer).

The resulting domain class diagram is presented in [4].

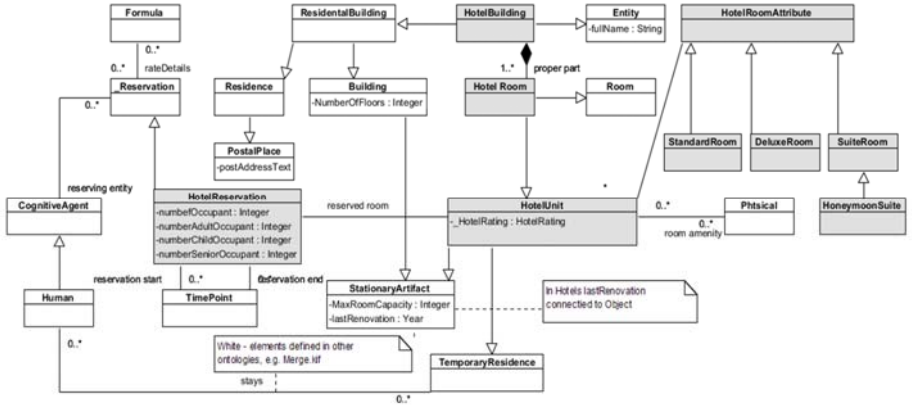

Figure 1. Hotel domain model elaborated based on SUMO ontologies

It can be noticed that, in the next step, the resulting class diagram can be simplified by domain expert without losing information, i.e. some classes can be removed while their properties are moved to specialization classes (e.g. *NumberOfFloors* attribute can be moved from *Building* class to *HotelBuilding* class what allows to remove *Building* class from the diagram).

5.2.6. Identified problems

Ontology, even if it is specific (like Hotel), contains many irrelevant elements. An expert has to decide what is contained in the scope of interest, and what can be omitted. For example Hotel ontology contains definitions of: 80 subclasses, 65 instances, 10 subrelations, 3 subattributes, from which we reused less than 20%.

The domain knowledge is spread over many ontologies – reading only one is not enough. Some facts are defined at very general level (predicates between *Object*, *Physical*) what makes the interpretation more difficult. Of course, that problem can be partially mitigated by simple refactorizations made at the end of domain model definition process.

Representation of some SUMO elements is difficult in the UML. For example, we had troubles in representation of SUMO attributes, especially when the relation is further refined (e.g. *SuiteRoom* which is a *subAttribute* of *HoneymoonSuite*). Because of that we decided to represent *HotelRoomAttribute* as a separate class (not a field in *HotelUnit* class).

6. Conclusion

The paper has concentrated on the problem of domain modeling in the context of ontology. Ontologies are a promising base for knowledge transfer from project to project in a certain application domain and from one development cycle of a project to the next. In the phase of system analysis, developers together with domain experts may use ontologies and transfer needed knowledge for the software application from domain being considered. Analysis of

a problem domain can be realized as an additional process of exploration of the ontology using existing tools. This approach is recommended if the domain knowledge of analysts is not sufficient to properly define a domain model, for example in case of difficulties in communicating with prospective users of the system. Taking into account also the aspect of performance, it should be developed a method for automatic transformation of required fragments of ontology directly to the model domain (currently this problem belongs to our interests and research). Proposed approach will guarantee to obtain domain models that meet the quality criteria (consistency, completeness, performance etc.).

Due to lack of space, the mappings between SUMO constructs and UML elements are presented only informally. The main contribution of the paper is an outline of a systematic approach to domain model definition.

We believe that such an approach may bring many benefits. The most important is to achieve a complete or high compliance with the modeled application domain. It is also reasonable to expect greater efficiency of the process of CIM construction.

References

[1] Hesse W., Ontologies in the Software Engineering Process, in: R. Lenz at al. (ed.): EAI 2005 – Proceedings of the Workshop on Enterprise Application Integration, 2005.

[2] Gašević D., Djurić D., Devedžić V., *Model Driven Architecture and Ontology Development*, Springer 2006.

[3] Happel H-J., Seedorf S., Applications of ontologies in software engineering, *Proc. of Workshop on Sematic Web Enabled Software Engineering (SWESE) on the ISWC*, 2006.

[4] Kleshchev A.S., How can ontologies contribute to software development?, *Knowledge Processing and Data Analysis*, Springer, Berlin, Heidelberg, 2011, 121-135.

[5] Mavetera N., Kroeze J., An ontology-driven software development framework, *Business Transformation through Innovation and Knowledge Management: An Academic Perspective*, Proc. of the 14th International Business Information Management Association Conference, IBIMA, 2010, 1713-1724.

[6] Gailly F., Geert P., Conceptual modeling using domain ontologies: Improving the domain-specific quality of conceptual schemas, *OOPSLA Workshop on Domain-Specific Modeling*, 2010, 109-114.

[7] Sugumaran V., Storey V., The role of domain ontologies in database design: An ontology management and conceptual modeling environment. *ACM Transactions on Database Systems (TODS)*, 31.3, 2006, 1064-1094.

[8] Tuzinkiewicz L., Sekulska-Grulich I., Modelowanie danych z wykorzystaniem ontologii, *Studia Informatica*, Vol. 33, Nu 2A, Gliwice, 2012, 365-377.

[9] Uzdanaviciute V., Butleris R., Ontology-based Foundations for Data Integration, in: *BUSTECH 2011, The First International Conference on Business Intelligence and Technology*, 2011, 34-39.

[10] Ontology Definition Metamodel, OMG Document Number: formal/2009-05-01, 2009.

[11] http://www.cyc.com/platform/opencyc

[12] http://suo.ieee.org/IFF/versions/20020515/IFFFoundationOntology.htm

[13] http://sigmakee.cvs.sourceforge.net/viewvc/sigmakee/KBs/engineering.kif

[14] Von Halle B., Business Rules Applied – Building Better Systems Using the Business Rules Approach, Wiley, 2002.
[15] Semantics Of Business Vocabulary And Rules (SBVR), Version 1.2, OMG Document, 2013.
[16] http://www.ontologyportal.org/

Presented at the 16th KKIO Software Engineering Conference, 22-24 September 2014, Poznań, Poland