INFORMATICA, 2012, Vol. 23, No. 3, 369–390

Application of the Ontology Axioms for the Development of OCL Constraints from PAL Constraints

Diana KALIBATIENĖ, Olegas VASILECAS

Vilnius Gediminas Technical University, Information System Research Laboratory, Information System Department
Saulėtekio al. 11, LT-10223, Vilnius, Lithuania
e-mail: diana@isl.vgtu.lt, olegas@isl.vgtu.lt

Received: December 2010; accepted: June 2012

Abstract. Nowadays, ontologies play a central role in many computer science problems such as data modelling, data exchange, integration of heterogeneous data and models or software reuse. Yet, if many methods of ontology based conceptual data modelling have been proposed, only few attempts have been made to ontology axioms based modelling of business rules, which make an integral part of each conceptual data model. In this paper, we present the approach how ontology axioms can be used for business rules implementation. Our proposal we apply for the transformation of PAL (Protege Axiom Language) constraints (ontology axioms), which is based on KIF (Knowledge Interchange Format) and is part of KIF ontology, into OCL (Object Constraint Language) constraints, which are part of a UML class diagram. Z language is used to formalise the proposal and describe the transformation. The Axiom2OCL plug-in is created for automation of the transformation and a case study is carried out.

Keywords: ontology axiom, conceptual data model, OCL constraint, UML class diagram, transformation, Z language.

1. Introduction and Problem Statement

Ontologies in nowadays are widely used for knowledge based data modelling, data exchange, integration of heterogeneous data and models or software reuse, since they are suitable to represent domain knowledge. Moreover, as stated in Jarrar *et al.* (2003), Guarino (1998), Wand *et al.* (1999), the semantic content expressed by the ontology can be transformed into information system (IS) components and, thus, reduce the costs of a conceptual modelling and assure the ontological adequacy of an IS. In most researches, like Guarino (1998), OMG (2005), Trinkunas and Vasilecas (2009), authors concentrate on ontology transformation into conceptual data model, like a UML class diagram or an entity-relationship diagram. However, there is not paid enough attention to ontology axioms and their usage for the modelling and implementing business rules, which are an integral part of each conceptual data model.

On the other hand, although, there is a number of methods and approaches (some of them are discussed below) for modelling and implementing business rules, it is not a trivial problem. There is no standard technology, yet. Nowadays tools used for the development of IS are not extended and adapted enough for modelling and implementing business rules. For example, in MagicDraw[1] OCL is used for implementation of business rules and constraints. However, there is no suitable interface to facilitate the definition of OCL constraints. In our country (Lithuania) a well know project VeTIS[2] was conducted to develop VeTIS tool for the definition of Business Vocabularies and Business Rules using controlled natural language, which is a subset of English language, and for transformation of SBVR (Semantics of Business Vocabulary and Business Rules; OMG, 2008) specifications into UML class diagrams with OCL constraints[2] (Nemuraite *et al.*, 2010). However, while almost all required Business Vocabularies definition capabilities are available, Business Rules description tools still lack some important features. Authors of VeTIS tool recommend you to keep rules (as well as your business processes) as simple as possible – not only because of the immaturity of the VeTIS tool, but also for the sake of your design models, your software and your business. Future efforts of authors are aimed at extending VeTIS capabilities by including more types of business rules; creating a user assistant for defining them; allowing configuration of the tool; reusing existing Business Vocabularies; and, most importantly, closer relation of the VeTIS capabilities with software development methodologies.

PowerDesigner[3] is suitable for modelling structural rules (using integrity constraints, foreign keys, domains, checks) only. There is no mechanism for defining and validating dynamic rules.

In this paper, we propose to use domain ontology and its axioms for modelling and implementing business rules, since both domain ontologies and conceptual models are intended to capture knowledge about a certain subject domain. From a syntactic perspective, in both artefacts, domain knowledge is represented in a similar way, i. e. knowledge are expressed in terms of concepts, their properties, relationships among concepts, and rules (in ontologies – axioms), which constrain concepts and their relationships in some manner. Moreover, ontology axioms (and ontology as a whole) are typically expressed in a formal way using a particular language. Therefore, ontology axioms can be transformed into a specific type of business rules automatically.

In this paper the main attention is paid to the applying the earlier proposed in Vasilecas *et al.* (2009) ontology-based method for the development of OCL constraints from PAL constraints. The next goal of the paper is to present a formal description of the PAL into OCL transformation using a particular formal language. The rest part of the paper is structured as follows. Section 2 presents motivation of the research and reviews related work on OCL and PAL. Section 3 describes the PAL into OCL transformation. The Axiom2OCL plug-in, developed on the basis of the proposed method, and examples of the transformation of PAL constraints into the corresponding OCL constraints are presented in Sections 4 and 5 respectively. Section 6 concludes the paper.

[1] `http://www.magicdraw.com/.`

[2] `http://www.magicdraw.com/files/manuals/VeTISUserGuide.pdf.`

[3] `http://www.sybase.com/products/modelingdevelopment/powerdesigner.`

The previous papers of the authors were centred on the creation of the rule system not its theory. I.e., the mapping of axioms to rules are presented and based more on empirical knowledge. The present paper is oriented to the creation of theory of the development rules from axioms.

2. Motivation and Related Work

Since in the rest of paper we are going to discuss a development of OCL constraints from PAL (Protege Axiom Language) constraints, this section presents a motivation and related work.

OCL (OMG, 2003a) is proposed as a formal language to express dynamic rules, since UML diagrams are typically not refined enough to express those rules explicitly. While UML with OCL satisfy the requirements of expressiveness, preciseness and unambiguity, OCL does not have any graphical notation and thus does not account for easily comprehensible language. The same can be sayd about all OCL-based languages and methods, like Badawy and Richta (2002), CDM Rule-Frame environment (Boyd, 2001) used by Oracle[4] (Armonas and Nemuraite, 2009). The analysis of related work on OCL shows that it is widely used for: expressing constraints and/or queries (like in Bejaoui *et al.* (2010) OCL is used to control topological relations of objects in spatial databases; in Pardillo *et al.* (2010) – for OLAP querying on conceptual multidimensional models of data warehouses; in Choi *et al.* (2009) – for restricting the modeled objects in a query modelling of XML-GL) and for the future transformation into SQL triggers or queries (Armonas and Nemuraite, 2009; Siripornpanit and Leckcharoen, 2009; Heidenreich *et al.*, 2008) or other executable code, like Java (Hamie, 2006). Therefore, it is relevant to develop OCL constraints and to propose methods for facilitating their development.

2.1. *Business Rules Implementation as OCL Constraints*

A number of methods are proposed to develop OCL constraints. Here we review some of them. Those methods are classified according to the used approach as follows:

- *Defining from scratch* – using this approach OCL constraints are defined manually. As presented in Zacharias (2008) a large part of rule-based systems are created without any specific development process using textual editors (33%) and simple text editors or textual rule editors with syntax highlighting (28%). This way of developing OCL constraints is the heaviest, since it demands high-level knowledge of OCL. Moreover, the likelihood of errors is highest.
- *Using structured English* – using this approach structured English or other languages are used to define OCL constraints. In "*The Semantics of Business Vocabulary and Business Rules*" (SBVR) OMG (2008) proposes to use logical for-

[4] `http://www.oracle.com/technology/software/products/database/index.html`.

mulations of rules or logical rules, which provide a formal, abstract, language-independent syntax for capturing the semantics of a body of shared meaning. However, they do not define the basic patterns or templates for rule definitions. They just suggest which keywords should be used in rules and how rule expressions should look like. Cabot *et al.* (2010) propose an automatic transformation from UML into SBVR specifications to facilitate interact with the business people (in their own language) to refine and validate the information modelled in the conceptual schema. In Kleiner *et al.* (2009) the transformation of SBVR into UML class diagram is proposed. In Raj *et al.* (2008) the transformation of SBVR into a set of UML diagrams, which includes Activity Diagram, Sequence Diagram and Class Diagram, is proposed. However, nothing is said about OCL constraints. As was said about, VeTIS tool uses controlled natural language and created to transform SBVR specifications into UML class diagrams with OCL constraints. However, it should be extended as presented above.

- *Template based* – using this approach rule templates are used to define OCL constraints. The Ross (1997) method proposes specific constructs for each of the rules families. However, a big number of modelling constructs makes the language quite complicated. In von Halle (2002) rules are expressed by rule templates, which are combination of rule clauses. However, there is no way for their implementation. Finally, there is lack of templates for OCL definition. Nemuraite *et al.* (2008) discuss the possibilities of using UML diagrams supplemented with OCL for expressing different types of rules. They present research fragments for possibility of generating OCL constraints from UML diagrams. A template-based approach is going to be applied for rule implementation.
- *Using rule abstractions* – using this approach rule abstractions, like decision tables, decision trees, etc., are used to define OCL constraints. In Normantas *et al.* (2009) decision tables are used for definition of OCL invariants. However, this approach is under development, now. The OMG proposed standard "*Production Rule Representation*" (PRR; OMG, 2009) presents the way of presenting production rules in UML diagrams and gives possibility of applying rule abstractions for the definition of rules.
- *Using domain knowledge* – using this approach domain ontology with axioms can be used for the conceptual data modelling with constraints, since ontology and a conceptual data model have the same conceptualization, e.g., both has concepts, relationships and rules (in ontology – axioms). The analysis of ontologies, like SUMO[5], and ontology development tools, like Protégé[4], from the implementation perspective shows that axioms are implemented using a particular language, like PAL or SWRL. A number of authors, like Trinkunas and Vasilecas (2009) propose to generate UML class diagrams from ontologies. However, they do not pay enough attention to rules and constraints. Milanovic *et al.* (2006) present a meta-model-driven transformation approach for sharing rules between OWL/SWRL and UML/OCL. The solution is based on the REWERSE Rule Markup Language (R2ML), as a pivotal meta-model and the bi-directional transformations between

OWL/SWRL and R2ML and between UML/OCL and R2ML. The main benefit of such an approach (OWL/SWRL to R2ML and R2ML to UML/OCL) is that UML/OCL rules can be mapped into all other rule languages (e.g., Jess, F-Logic, and Prolog) that have mappings defined with R2ML. The implementation is done by using Atlas Transformation Language (ATL). However, authors present only a little part of the mapping between OWL/SWRL and R2ML and UML/OCL. An OWL/SWRL meta-model is not presented in the paper, also. Therefore, the proposed approach should be refined in the future.

2.2. *Why PAL Constraints?*

As a target language we choose PAL, which is used for writing strong logical constraints in Protege[5] ontologies. The analysis of existing domain ontologies *with axioms*, which can be used for the development of conceptual data models with constraints, shows that KIF (Knowledge Interchange Format), which is used as a basis for PAL, is the most popular for the description of ontologies. For example, SUMO[6] uses SUO-KIF[7] SUMO language, a part of Protege ontologies[4], use KIF or PAL for expressing axioms.

Though, nowardays Ontology Web Language (OWL; OMG 2005) with SWRL (the Semantic Web Rule Language; O'Connor, 2010), which is an OWL-based rule langue developed to express rules in terms of OWL concepts, become one of the most popular languages for expressing domain ontologies (Horrocks *et al.*, 2004; Milanovic *et al.*, 2006; Sirin and Tao, 2009). Therefore, another unsolved question is transforming KIF based ontologies *with axioms* into OWL/SWRL ontologies. In this paper authors are not discussing this problem.

2.3. *Choosing a Formal Language to Specify the PAL to OCL Transformation*

Formal specifications of systems have been a focus of software engineering research for many years and applied in a wide variety of settings. Their industrial use is still limited because of limitation of a graphical notation, understandability, lack of a tool support, and the cost of a specification. The specification language must first be learned, and then experienced before its using. Advantages of the usage of formal languages are formality, preciseness, reducing the likelihood of errors, exploration of design choices, and the quality of documentation.

We are going to use a particular state-based language (it characterises the admissible system states at some arbitrary snapshot) for the definition of the transformation of PAL constraints into OCL constraints, since it is necessary to define components of PAL constraints (or source state), components of OCL constraints (or target state) and their mapping. Z, VDM and B are the known and widely used state-based languages. Therefore, we have to choose among them.

[5] `http://protege.stanford.edu.`
[6] `http://www.ontologyportal.org/.`
[7] `http://suo.ieee.org/SUO/KIF/suo-kif.html.`

A number of authors discuss the differences and similarities, advantages and disadvantages of these languages. Especially there is popular comparison of Z with VDM (Hayes *et al.*, 1994; Spivey, 1988; Fensel, 1995). However, we can see that all these languages are suitable for the definition of different states of a system. Therefore, considering our knowledge of Z, we choose it for the definition of the transformation of ontology axioms into OCL constraints. Moreover, Z has been accepted as the ISO standard in 2002 (ISO, 2002). However, we are not discounting the possibility of using other formal language for the specification of the proposed transformation.

3. Transforming PAL Constraints into OCL Constraints

Based on the related work and our own experience (Vasilecas *et al.*, 2009), in this section we present transformation of ontology axioms into OCL constraints. The description of the method of the transformation of ontology axioms into business rules, which is applied in this paper, is presented in Vasilecas *et al.* (2009). Here we present only the transformation of ontology axioms, presented in PAL, into OCL constraints.

In this section the definition of elements of a UML class diagram and Protégé ontology are presented to ensure the completeness of the transformation.

3.1. *Definition of a Protégé Ontology Using Z*

According to Knublauch (2006), a *ProtégéOntology* schema is defined using Z from the smallest parts in the following way. First, the definition of a slot, presenting property of a class, in the *StandardSlot* schema (Fig. 1) is proposed. Syntactically, a slot has a name, a given type and documentation, which consists of a set of characters. A set *Name* is defined to present the names for all classes and the names of all slots. A set *PType* denotes all possible types in Protégé ontology. They are any, Boolean, float, instance, integer, string and symbol. Symbol is an enumerated list of slot values, like red, blue, green. Instance is a type of a slot, whose value is the instance of a class. A partial function

$$[Name, PType]$$

$$PType ::= any \mid boolean \mid float \mid instance \mid integer \mid string \mid symbol$$

StandardSlot

$$slotname : Name$$

$$slotvaluetype : PType$$

$$docum : \mathbb{P}\,(\mathbb{Z} \times \mathrm{CHAR})$$

$$slotstate : slotname \nrightarrow slotvaluetype$$

$$slot = \mathrm{dom}\ slotstate$$

Fig. 1. The *StandardSlot* schema defined in Z.

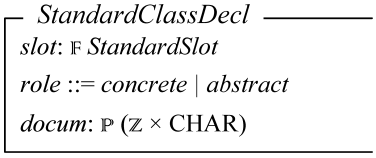

Fig. 2. The *StandardClassDecl* schema defined in Z.

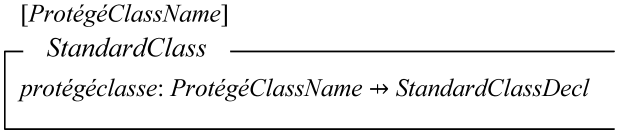

Fig. 3. The *StandardClass* schema defined in Z.

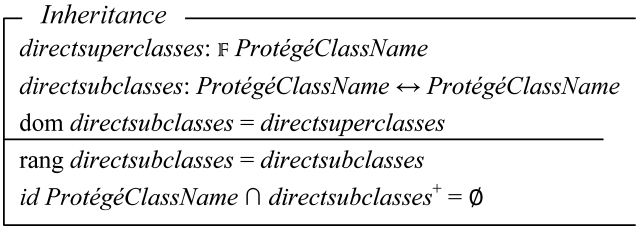

Fig. 4. The *Inheritance* schema defined in Z.

slotstate is declared to map slots to their types. A constraint presented in the bottom part of the schema denotes that the number of slots equals to the domain[8] of *slotstate*.

The main component of Protégé ontology is a Protégé class, which is an abstract representation of a concept in an application domain. A Protégé class has slots. A class is defined in a *StandardClassDecl* schema, which consists of: a finite set of slots, a role, which is an abstract or a concrete, and documentation, which is a description of a class (Fig. 2).

Names of Protégé classes should be unique in the enclosing name space. Therefore, the set of Protégé classes is defined as a partial function from *PClassName* to *Standard-ClassDecl* (Fig. 3).

In Protégé, generalization is used to express a taxonomic relationship between classes. Therefore, two variables, *directsupeclasses* and *directsubclasses*, are defined to express relationships between classes involved in generalizations (Fig. 4). The variable *direct-superclasses* is defined as a finite state of *ProtégéClassName* denoting all superclasses. The variable *directsubclasses* is defined as a relation between values of type *Protégé-ClassName*. Its domain is the set of superclasses and its range[9] is the set of subclasses. A class cannot be a superclass of itself or any of its ancestors.

[8] In the Z notation (Bowen, 2003), the domain of a relation R: $T \leftrightarrow U$ is the set of all elements in T which are related to at least one element in U by R.

[9] In the Z notation (Bowen, 2003), the range of R is the set of all elements in U which are related to at least one element in T by R.

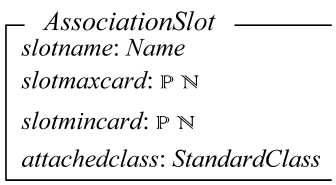

Fig. 5. The *AssociationSlot* schema defined in Z.

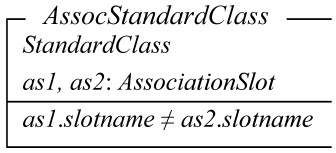

Fig. 6. The *AssocStandardClass* schema defined in Z.

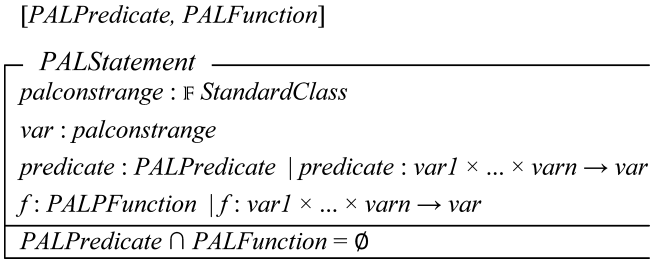

Fig. 7. The *PALStatement* schema defined in Z.

In Protégé, a standard allowable relationship between classes is a binary relationship. Therefore, we discuss only binary relationships. In Protégé, binary relationships are presented by slots, which type is an instance. Such a slot, an association slot, defines a binary relationship between a class, to which it belongs, and a class, which is indicated as an attached class. Syntactically, each association slot has a name, which denotes the name of an association, an instance type, which denotes a class, to which a given class is associated, *slotmaxcard*, which denotes a maximum cardinality of a slot, *slotmincard*, which denotes minimum cardinality of a slot. An *AssociationSlot* schema presents the components of an association slot (Fig. 5). A slot cardinality maps to the whole infinite non-negative integer set, which is represented as $\mathbb{N}$ in Z.

An *AssocStandardClass* schema presents a Protégé class, which has an association slot (Fig. 6). A Protégé class can have several association slots; however, their names should differ.

Now, we are going to define axioms. In Protégé, axioms are implemented by PAL constraints. Syntactically, a PAL constraint consists of a name, a description, a range, which is a local or a global variable that appear in a constraint, and a PAL statement (Fig. 8). A PAL statement (Fig. 7) is composed of a set of embedded predicates and functional expressions that involve variables ranging over a set of values (over a range), and that ultimately return true or false on each particular instantiation of variables. Therefore,

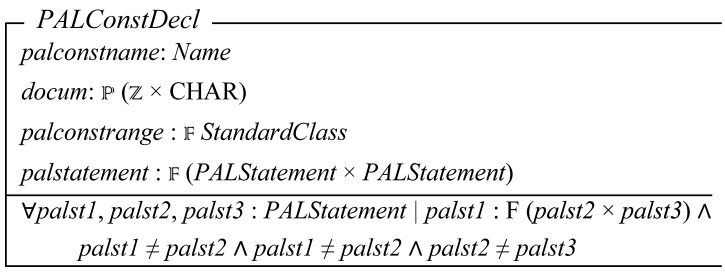

Fig. 8. The *PALConstDecl* schema defined in Z.

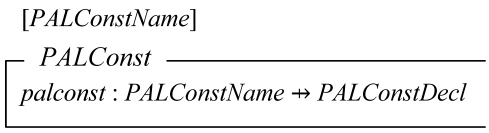

Fig. 9. The *PALConst* schema defined in Z.

we define two sets, *PALPredicate*, from which the predicates for PAL statements can be taken, and *PALFunction*, which denotes PAL functions. For more about predicates and functions you can find in Crubézy (2005).

As was described in Vasilecas *et al.* (2009), axioms and consequently PAL constraints are of the form *state* or *condition-state*. A *state axiom* clearly defines a state in which a domain should be. A *condition-state axiom* defines an admissible state of a domain under the defined condition. Therefore, as presented in Fig. 8, a PAL statement can have two parts (condition and state), which are also statements. However, it is important to note that if a PAL statement consists of other PAL statements, which are predicates or functions, these statements are not equal. E.g., there is no reflexive relationship between PAL statements.

Names of PAL constraints should be unique in the enclosing name space. Therefore, the set of PAL constraints is defined as a partial function from *PALConstName* to *PALConstDecl* (Fig. 9).

3.2. *Definition of a UML Class Diagram Using Z*

We apply the definition of a UML class diagram using Z from Kim and Carrington (1999). Since it varies little from the original definition, we present it in Annex 1.

3.3. *Definition of an OCL Constraint Using Z*

Syntactically, an OCL constraint has a context, an expression type, a name, which is unique for all constraints, and an expression (OMG, 2003a). Therefore, we are going to analyse these components.

OCL is a strongly typed language (OMG, 2003a). A type is assigned to every OCL expression and typing rules determine in which ways well-formed expressions can be constructed.

Type

$Type$

$\omega \in \Omega \mid t_1, ..., t_n \in Type \wedge t \in Type \wedge \omega : t_1 \times \cdots \times t_n \rightarrow t$

Fig. 10. The *Type* schema defined in Z.

DataSignature

$Type_M \in Type$

$\leq$

$\Omega_M \in \Omega$

Fig. 11. The *DataSignature* schema defined in Z.

Types are associated with a set of operations, which are functions over values of the type. The semantics of types and operations is defined by a mapping that assigns each type a domain and each operation a function. The meaning of the type stays the same as in the previous section (e.g., all possible types in UML). The syntax of operations (ω) is presented in Fig. 10.

An operation (ω) is described by defining its name, the parameter types ($t_1 \times \cdots \times t_n$), and the result type ($t$). An example of operations are arithmetic operations $+$, $-$, $\_$, /, etc. for integers and real numbers, division (div) and modulo (mod) of integers, sign manipulation ($-$, abs), conversion of *Real* values to *Integer* values (floor, round), and comparison operations ($<$, $>$, $<$, $>$).

A data signature over an object model consists of types used in this model, a type hierarchy ($\leqslant$) over types and operations over types (Fig. 11).

The type hierarchy is defined for all types like (for more see (OMG, 2003a)):

- $\leqslant$ is (a) $t \leqslant t$ – reflexive, (b) $t'' \leqslant t'\ t' \leqslant t \Rightarrow t'' \leqslant t$ – transitive, and (c) $t'' \leqslant t'\ t \leqslant t \Rightarrow t = t'$ – asymmetric.
- *Integer* $\leqslant$ *Real*, etc.

OCL itself is an expression language (OMG, 2003a). The definition of expressions is based upon the data signature (Fig. 11), which contains the initial set of syntactic elements upon which the expression syntax is built. For each expression the set of variables (*Var*), which are indexed by a type t, is also defined. An expression can be understood as a function on some variables of a specific type (Fig. 12). The details about possible functions can be seen in OMG (2003a) and not presented here, since we need to define the structure of an OCL constraint only. Some functions can be defined by users, using previously defined operations on variables.

An OCL expression is always written in some syntactical context. The UML model (in our case a UML class diagram) itself provides the most general kind of context. The *context* can be a particular class, an attribute, a method or other element of a UML class diagram.

An expression type (*exprtype*) is a possible type of an OCL expression. It can be a stereotype (invariant, precondition or postcondition), an initial value, which is used to

Expression

$DataSignature$

$Var_t \in Var \mid t \in Type_M$

$Expr_t \in Expr \mid t \in Type_M \land Expr \rightarrow F\,(Var) \rightarrow \mathbb{P}\,(\Omega_M(Var))$

Fig. 12. The *Expression* schema defined in Z.

OCLConstraint

$Context \in UMLClassDiagram$

$name\colon \mathbb{F}\ Name \mid name \nrightarrow OCLConstraint$

$exprtype ::= inv \mid pre \mid post \mid init \mid derive$

$expression\colon \mathbb{P}\ (Expression)$

Fig. 13. The *OCLConstraint* schema defined in Z.

represent the initial value in an OCL expression, and derived value, which is used to represent a derivation rule. An *invariant* is an expression with Boolean result type and a set of (explicitly or implicitly declared) free variables. These variables are declared by a context. A system state is called valid with respect to an invariant if the invariant evaluates to true.

Pre- and *postconditions* are used to define the system state before the execution of an operation and the system state that results from the operation execution. Parameters (variables) used in pre- and postconditions are declared by a context.

The keyword *self* can be used to denote variables from the context.

An example of an expression is *self.numberOfEmployees* > 50, where *numberOf Employees* is an attribute (a variable) and 50 is a possible value of this attribute (a variable).

Finally, an OCL constraint defined in Z is presented in Fig. 13.

3.4. *Definition of the Transformation of PAL Constraints into OCL Constraints Using Z*

Since the definition of Protégé ontology with PAL constraints and a UML class diagram with OCL constraints is prepared, we can define the transformation of PAL constraints into OCL constraints.

A *PALtoOCL* schema (Fig. 14) defines the transformation of PAL constraints into OCL constraints. The *PALConst* and *OCLConstraint* schemas are included to the *PALtoOCL* schema, since their variables are used to define the transformation. A name of a PAL constraint is transformed into the corresponding name of an OCL constraint. A range of a PAL constraint is transformed into the corresponding context of an OCL constraint. A PAL statement is transformed into the corresponding OCL expression.

It is difficult to define a type of an OCL expression. The analysis of PAL shows that all PAL statements correspond to OCL invariants. Therefore, an expression type is equated to invariant during the transformation of PAL constraints into OCL constraints.

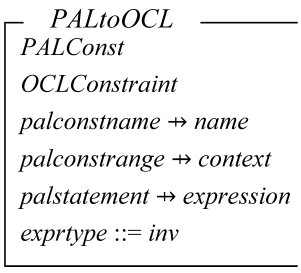

Fig. 14. The *PALtoOCL* schema defined in Z.

Table 1
Mapping Protégé ontology element to UML class diagram elements

Elements of Protégé ontology	Elements of UML class diagram
Protégé ontology	UML class diagram
"Thing"	Class
Class	Class
Slot	Attribute
Documentation	*Comment*
Value types:	Data types:
Any	Not defined
Boolean	Boolean
Glass	Relationship with appropriate class
Float	Float
Instance	Relationship with appropriate class
Integer	Int
String	Char
Symbol	Enumeration
Required	Multiplicity: 1
Minimum	Multiplicity:
Maximum	Multiplicity:
Default values	Default value
Is-a relation (directed-binary-relation)	Generalization
PAL constraint	OCL constraint

For more details about mapping of the Protégé ontology to a UML class diagram can be seen in Knublauch (2006) and in Table 1.

The main steps of the proposed transformation are presented in Fig. 16.

An example of the mapping of a PAL constraint to an OCL constraint follows:

- This part of a PAL statement defines that a value of a slot `start_date` of the class Employee should be less than a value of a slot `end_date` of the same class.

```
(< ( 'start_date' ?Employee) ('end_date' ?Employee)))
```

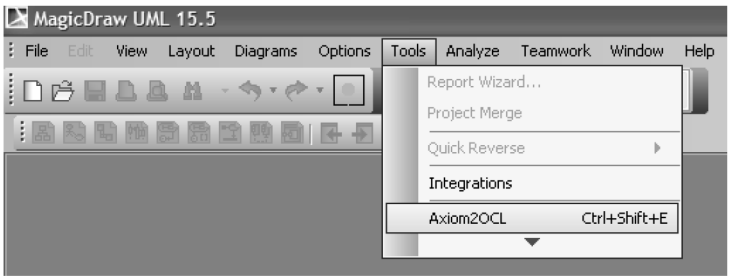

Fig. 15. The *Axiom2OCL* plug-in.

- The presented PAL constraint corresponds to an OCL constraint. ?Employee is transformed into the context of an OCL constraint. All slots of the PAL statement are transformed into the corresponding attributes in the OCL constraint. For example, "`end_date`" is transformed into "`self.end_date`".

```
context Employee inv:
self.end_date > self.start_date
```

Table 1 presents the mapping of Protégé ontology UML class diagram. It is used as a basis to define the mapping of PAL constraints to OCL constraints. *Note* that Table 1 is not the final version of the mapping Protégé ontology to UML class diagram.

4. *Axiom2OCL* – The Plug-in for Transforming PAL Constraints into OCL Constraints

The prototype of the *Axiom2OCL* plug-in is created to implement the proposed method of the transforming PAL constraints into OCL constraints and to support our statement that ontology axioms could be used for the development of business rules. The plug-in is developed according to the proposed the *PALtoOCL* schema.

The plug-in can be attached to *MagicDraw UML 15.5*[10] or *Protégé 3.0* (or other version). Figure 15 presents the *Axiom2OCL* plug-in attached to MagicDraw UML 15.5[10].

Figure 16 presents the main steps of the developed *Axiom2OCL* plug-in, which is based on the *PALtoOCL* schema. In this prototype the user should denote the input file, in which PAL constraints are stored, and may denote the output file, where OCL constraints will be stored. If user does not denote the output file, the plug-in creates a default output file. After the denoting the input and output files, all PAL constrains from the input file are automatically transformed into OCL constraints.

The plug-in is created in the Java development environment.

[10] `https://www.magicdraw.com/`.

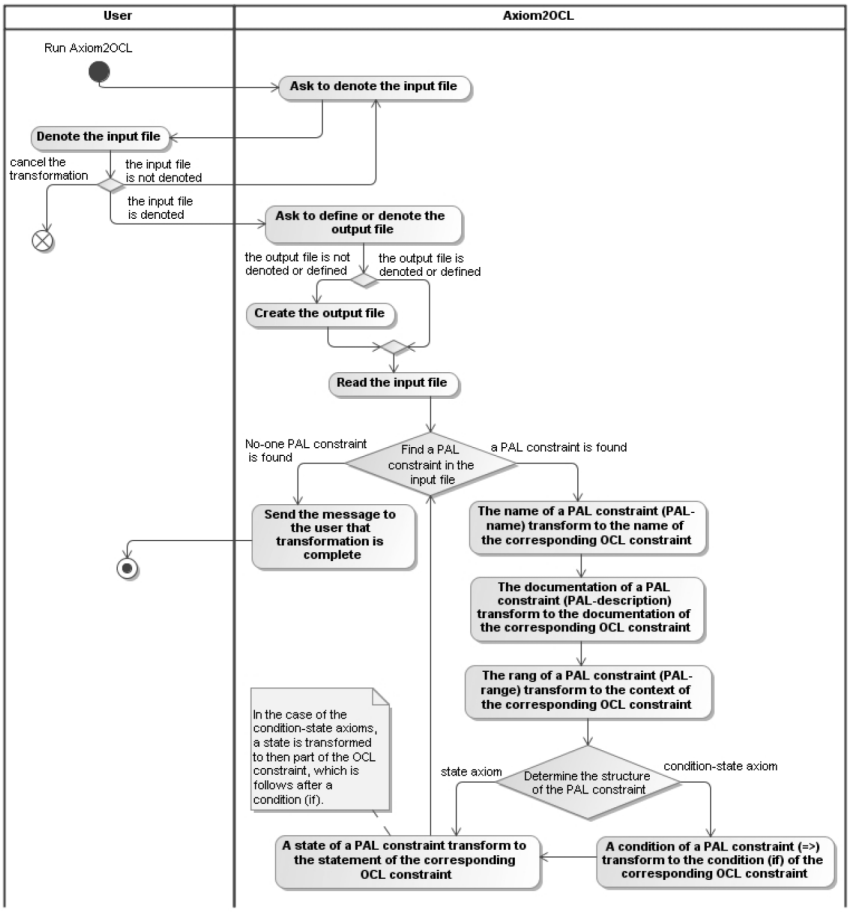

Fig. 16. The main steps of *Axiom2OCL*.

5. An Example of Transforming a PAL Constraint into an OCL Constraint

An example of the transformation of PAL constraint, restricting that *the Employee end date should be after the start date*, into the corresponding OCL constraint, represented as follows:

- A PAL constraint:

```
(%3APAL-NAME ”editor-employees-salary-constraint”)
(%3APAL-RANGE ”(defrange ?editor :FRAME Editor)\ n(defrange
?employee :FRAME Employee responsible_for)”)
(%3APAL-STATEMENT ”(forall ?editor (forall ?employee\ n (=>
(and \ n (responsible_for ?editor ?employee)\ n (own-slot-
not-null salary ?editor) \ n (own-slot-not-null salary ?em-
ployee)) \ n (> (salary ?editor) (salary ?employee)))))”))
```

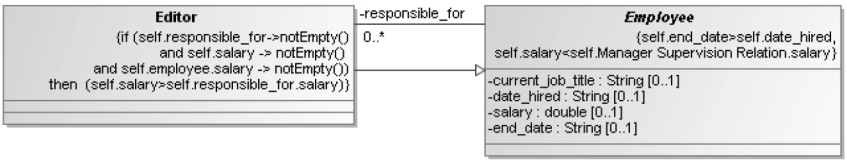

Fig. 17. The part of a newspaper class diagram.

- A corresponding OCL constraint:
```
context Editor inv editor-employees-salary-constraint:
IF (self.responsible_for->notEmpty() AND self.salary ->
notEmpty() AND self.employee.salary -> notEmpty())
THEN (self.salary>self.responsible_for.salary) endif
```

The corresponding OCL constraint is attached to the part of a newspaper class diagram (Fig. 17), which is generated from the newspaper ontology using UMLBackend plug-in (Knublauch, 2006).

Conducting an experiment, 10 PAL constraints were transformed into OCL constraints. At this moment the prototype is suitable for the development of OCL invariants (a class of dynamic constraints) from PAL constraints only, since in Protégé there is no difference between constraints and derivation axioms. Therefore, specific keywords for PAL constraints or other approach should be used for the development of other OCL constraints, e.g., not invariants.

In the future the proposed method and prototype should be refined and adapted for the transformation of more complex constraints, like derivation axioms to derivation rules.

The proposed transformation of PAL constraints into OCL constraints is applied in the High Technology Development Program Project "Business Rules Solutions for Information Systems Development (VeTIS)"[11] for extending MagicDraw tool to generate OCL constraints from PAL constraints.

6. Conclusions

The analysis of the related works in the field of knowledge-based information systems development using the domain ontology shows that business rules are presented in the ontology by axioms and defined using ontology concepts. However, those axioms are not used for the development of business rules, mostly. Yet OCL is widely used for expressing constraints and/or queries, there is no graphical notation facilitating their development.

The comparative analysis of approaches used for definition of OCL constraints shows that domain knowledge presented by ontology can be used here. Therefore, the authors of this paper propose the transformation of ontology axioms, presented in PAL, into OCL constraints. The proposed transformation is defined by Z notation.

[11] `http://www.verslotaisykles.lt/VeTIS/.`

A created plug-in *Axiom2OCL* and was used to carried out the experiment show that the proposed transformation is suitable for automatic generation of OCL invariants and can be extended for the generation of derivation constraints.

ANNEX 1. Definition of a UML Class Diagram Using Z

In UML, a class is a type that has objects as its instances (OMG, 2003b). Syntactically, a class has a name, attributes and operations and participates in inheritance hierarchies. Attributes have names and types. Operations have names and parameters. Each parameter of an operation has a name and a given type. Kim and Carrington (1999) define two given sets, *ClassName* and *Name*, from which the names of all classes and the names of all attributes, operations and operation parameters can be drawn, respectively. *Type* is a meta-type that is portioned into all possible types in UML. A *ClassDecl* schema denotes the components of a class: a finite set of attributes and operations (Fig. 18). A partial function *attrstate* is used to map attributes to their types. A partial function *opsigs* is used to map operations to their parameters and also to map each parameter to its type.

Class names should be unique in the enclosing name space. Thus, the set of classes is defined as a partial function from *ClassName* to *ClassDecl* (Fig. 19).

In UML relationships between classes are represented as associations. Associations can be: common association, aggregation and composition. In most cases, binary associations are used in a class diagram. Moreover, aggregation and composition are always binary relationships. Therefore, only binary associations are presented here. Syntactically a binary association is represented as a link between two classes with an association name and two association ends. Each association end has a role name, cardinality and a class to which the association end is attached. For aggregation and composition, an aggregation indicator, is added to one of the association ends.

$$[ClassName, Name, Type]$$

ClassDecl

$$attribute : \mathbb{F}\ Name$$
$$operation : \mathbb{F}\ Name$$
$$attrstate : Name \pfun Type$$
$$opsigs : Name \pfun (Name \pfun Type)$$

$$attribute = \mathrm{dom}\ attrstate$$
$$operation = \mathrm{dom}\ opsigs$$

Fig. 18. The *ClassDecl* schema defined in Z (Kim and Carrington, 1999).

UMLClass

$$classes : ClassName \pfun ClassDecl$$

Fig. 19. The *UMLClass* schema defined in Z (Kim and Carrington, 1999).

$$assockind ::= none \mid aggregate \mid composite$$

AssociationEnd

$$rolename: Name$$
$$cardinality: \mathbb{P}\ \mathbb{N}$$
$$attachedclass: ClassName$$
$$aggregation: assockind$$

$$cardinality \neq \{0\}$$

Fig. 20. The *AssociationEnd* schema defined in Z (Kim and Carrington, 1999).

UMLAssociation

$$associations: \mathbb{P}\ (Name \times (AssociationEnd \times AssociationEnd))$$

$$\forall n: Name; e1, e2: AssociationEnd \mid (n, (e1, e2)) \in associations \bullet$$
$$e1.rolename \neq e2.rolename \neq n$$
$$e1.aggregation \in \{aggregate, composite\} \Longrightarrow e2.aggregation = none$$
$$e1.aggregation = composite \Longrightarrow e1.multiplicity = \{1\}$$
$$\forall n1, n2: Name; e1, e2, e3, e4: AssociationEnd \mid \{e1, e2\} \neq \{e3, e4\} \wedge$$
$$\{(n1, (e1, e2)), (n2, (e3, e4))\} \subseteq associations \bullet$$
$$\{e1.attachedclass, e2.attachedclass\} = \{e3.attachedclass, e4.attachedclass\} \Longrightarrow$$
$$n1 \neq n2$$

Fig. 21. The *UMLAssociation* schema defined in Z (Kim and Carrington, 1999).

An *AssociationEnd* schema denotes the components of an association end: a role name, cardinality and an attached class (Fig. 20). A cardinality in UML denotes the number of allowable instances that may be associated with a single instance of the class attached to its opposite association end. A cardinality is a sequence of non-negative integer intervals in the format *lower-bound . . . upper-bound*, where the upper bound can be unlimited (denoted by star symbol *). A variable *cardinality* is a set of non-negative integer values. When cardinality comprises the star symbol *, it maps to the whole infinite non-negative integer set, which is represented as $\mathbb{N}$ in Z. The variable *aggregation* denotes whether or not the attached class is an aggregate. This variable can take the values *none*, *aggregate*, or *composite*. The constraint in the predicate part states that cardinality cannot have the value zero for both its lower and upper bounds.

An association can appear more than once in a class diagram and a binary association has exactly two association ends. The set of binary associations is defined as a power set of tuples of *Name* and a pair of *AssociationEnd.ei* is a composite or aggregate.

Figure 21 presents the *UMLAssociation* schema defined in Z.

The constraints for the *UMLAssociation* schema state that:

- an association name must be different from both role names and each role name also must be different;
- for aggregation and composition, there should be an aggregate or a composite end and the other end and should have the aggregation value of none;

AssocClassDecl

ClassDecl
e1, *e2*: *AssociationEnd*

$e1.rolename \neq e2.rolename$
$e1.aggregation = none$
$e2.aggregation = none$

Fig. 22. The *AssocClassDecl* schema defined in Z (Kim and Carrington, 1999).

AssocClassDecl

ClassDecl
e1, *e2*: *AssociationEnd*

$e1.rolename \neq e2.rolename$
$e1.aggregation = none$
$e2.aggregation = none$

Fig. 23. The *AssocClassDecl* schema defined in Z (Kim and Carrington, 1999).

UMLAssocClass

$assocClasses: ClassName \nrightarrow AssocClassDecl$

$\forall c: ClassName; ac: AssocClassDecl \mid c \mapsto ac \in assocClasses \bullet$
$c \notin \{ac.e1.attachedclass, ac.e2.attachedclass\}$
$ac.attribute \cap \{ac.e1.rolename, ac.e2.rolename\} = \emptyset$

Fig. 24. The *UMLAssocClass* schema defined in Z (Kim and Carrington, 1999).

- for composition, the multiplicity of the composite end must equal one;
- all associations must have a unique combination of name and associated classes. Thus, if attached classes are the same, their association names should be different.

Association classes are similar to associations except that they have class-like properties in terms of attributes and operations. That is, association classes have properties of both classes and associations. The *AssocClassDecl* schema (Fig. 22) inherits *ClassDecl* and includes two association ends. The constraints define that the two role names must be different and the *aggregation* value of both association ends is *none*.

The set of association classes is declared as a partial function from *ClassName* to *AssocClassDecl* (Fig. 23).

The constraints describe well-formed rules for association classes (Fig. 15): an association class cannot be defined between itself and something else and; the role names and the attribute names do not overlap.

In UML generalization is the taxonomic relationship between objects, in which objects of the superclass have general information and objects of the subclasses have more specific information (OMG, 2003b). Two variables, *superclasses* and *subclasses* are declared to express relationships between classes involved in generalizations and constraints on them (Fig. 24). The variable *superclasses* is defined as a finite state of *ClassName* de-

UMLGeneralization

$superclasses: \mathbb{F}\ ClassName$

$subclasses: ClassName \leftrightarrow ClassName$

$\text{dom}\ subclasses = superclasses$

$\text{rang}\ subclasses = subclasses$

$id\ ClassName \cap subclasses^{+} = \emptyset$

Fig. 25. The *UMLGeneralization* schema in Z (Kim and Carrington, 1999).

UMLClassDiagram

UMLClass

UMLAssociation

UMLAssocClass

UMLGeneralization

$\text{dom}\ (assocClasses) \cap \text{dom}\ (classes) = \emptyset$

$\forall n; Name; e1, e2; AssociationEnd \mid (n, (e1, e2)) \in associations \bullet$
$\{e1.attachedclass, e2.attachedclass\} \subseteq \text{dom}\ (classes) \cup \text{dom}\ (assocClasses)$
$e1.rolename \notin classes\ (e2.attachedclass).attribute$
$e2.rolename \notin classes\ (e1.attachedclass).attribute\}$

$\forall n: \text{dom}\ (assocClasses); e1, e2: AssociationEnd \mid$
$e1 = assocClasses(n).e1 \wedge e2 = assocClasses(n).e2 \bullet$
$\{e1.attachedclass, e2.attachedclass\} \subseteq \text{dom}\ (classes) \cup \text{dom}\ (assocClasses)$
$e1.rolename \notin classes\ (e2.attachedclass).attribute$
$e2.rolename \notin classes\ (e1.attachedclass).attribute\}$

$\text{dom}\ (subclasses) \cup \text{rang}\ (subclasses) \subseteq \text{dom}\ (classes) \cup \text{dom}\ (assocClasses)$

Fig. 26. The *UMLClassDiagram* schema defined in Z (Kim and Carrington, 1999).

noting all superclasses. The variable *subclasses* is defined as a relation between values of type *ClassName*. Its domain is the set of superclasses and its range is the set of subclasses. A class cannot be a superclass of itself (reflexive inheritance) or any of its ancestors.

Finally, a UML class diagram is a collection of classes including classes in generalizations and association classes, and associations between these classes (Fig. 25).

The constraints describe that:

- classes and association classes are disjoint;
- classes that are involved in associations or association classes should be classes in the diagram;
- for an association or an association class, the role name at an association end should be different from the attribute names of the class attached to the other end;
- classes involved in generalizations should be classes in the diagram.

References

Armonas, A., Nemuraite, L. (2009). Using attributes and merging algorithms for transforming OCL expressions to code. *Information Technology and Control*, 38(4), 283–293.

Badawy, M.; Richta, K. 2002. Deriving triggers from UML/OCL specification. In: Kirikova, M. (Ed.), *Information Systems Development: Advances in Methodologies, Components and Management,* Kluwer Academic/Plenum, New York, pp. 305–316.

Bejaoui, L., Pinet, F., Schneider, M., Bedard, Y. (2010). OCL for formal modelling of topological constraints involving regions with broad boundaries. *Geoinformatica*, 14(3), 353–378.

Bowen, J. (2003). *Formal Specification and Documentation using Z: A Case Study Approach*. International Thomson Computer Press (ITCP).

Boyd, L. (2001). *CDM RuleFrame – The Business Rule Implementation Framework That Saves You Work*. Oracle Corporation, iDevelopment Center of Excellence document. `http://www.dulcian.com/papers/ODTUG/2001/BR%20Symposium%202001/Boyd_BR.htm` [2008 11 10].

Cabot, J., Pau, R., Raventos, R. (2010). From UML/OCL to SBVR specifications: a challenging transformation. *Information systems*, 35(4), 417–440.

Choi, B.J., Kim, Y.S., Ha, Y. (2009). UML for XML-GL query using class diagram and OCL. In: Lee, R., Du, W., Kim, H.K., Xu, S. (Eds.), *Proc. of the 7th ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2009)*. IEEE, New York, pp. 179–85.

Crubézy, M. (2005). *Supported Predicates and Functions in PAL*. `http://protege.stanford.edu/plugins/paltabs/pal-quickguide/PalKeywords/pal-predicates-functions.html` (February 2010).

Fensel, D. (1995). Formal specification languages in knowledge and software engineering. *The Knowledge Engineering Review*, 10(4), 361–404.

Guarino, N. (1998). Formal ontology and information systems. In: *Proc. of FOIS'98*. IOS Press, pp. 3–15.

Hamie, A. (2006). On the relationship between the Object Constraint Language (OCL) and the Java Modeling Language (JML). In: *Proceedings of the 7th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pp. 411–414.

Hayes, I.J., Jones, C.B., Nicholls, J.E. (1994). Understanding the differences between VDM and Z. *ACM SIGSOFT Software Engineering Notes*, 19(3), 75–81.

Heidenreich, F., Wende, C., Demuth, B. (2008). A framework for generating query language code from OCL invariants. In: *Proceedings of the 7th OCL Workshop at the UML/MoDELS Conferences*, EASST, Vol. 9. `http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/108/103` (May 2010).

Horrocks, I., *et al.* (2004). SWRL: A semantic web rule language combining OWL and RuleML. In: *W3C document*. `http://www.w3.org/Submission/SWRL/` (January 2009).

ISO (2002). *Information Technology – Z Formal Specification Notation – Syntax, Type System and Semantics*, ISO/IEC 13568:2002.

Jarrar, M., Demey, J., Meersman, R. (2003). On using conceptual data modeling for ontology engineering. In: Spaccapietra, S. *et al.* (Eds.), *Journal on Data Semantics*, *LNCS*, Vol. 2800. Springer, Berlin, pp. 185–207.

Kim, S.K., Carrington, D. (1999). Formalizing the UML class diagram using object-Z. In: Goos., G., Hartmanis, J., van Leeuwen, J. (Eds.), *Proceedings of the Standard Second International Conference Fort Collins, LNCS*, Vol. 1723. Springer, Berlin, pp. 83–98.

Kleiner, M., Patrick, A., Bezivin, J. (2009). Parsing SBVR-based controlled languages. In: *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems*. Springer, Berlin, pp. 122–136.

Knublauch, H. (2006). *UMLBackend*. Stanford Medical Informatics, Stanford University document. `http://protege.cim3.net/cgi-bin/wiki.pl?UMLBackend` (October 2006).

Milanovic, M., Gaševic, D., Giurca, A., Wagner, G., Devedžic, V. (2006). On interchanging between OWL/SWRL and UML/OCL. In: *Proceedings of 6th OCL Workshop at the UML/MoDELS Conference*, Genova, Italy, pp. 81–95.

Nemuraite, L., Ceponiene, L., Vedrickas, G. (2008). Representation of business rules in UML&OCL models for developing information systems. In: Stirna, J., Persson, A. (Eds.), *Proceedings of the Practice of Enterprice Modeling (PoEM 2008)*, Stockholm, Sweden. *Lecture Notes in Business Information Processing*, Vol. 15. Springer, Berlin, pp. 182–196.

Nemuraite, L., Skersys, T., Šukys, A., Šinkevicius, E., Ablonskis, L. (2010). VETIS tool for editing and transforming SBVR business vocabularies and business rules into UML&OCL models. In: Targamadze, A., Butleris, R., Butkiene, R. (Eds.), *Proceedings of the 16th International Conference on Information and Software Technologies (IT 2010)*. Technologija, Kaunas, pp. 377–384.

Normantas, K., Vasilecas, O., Sosunovas, S. (2009). Augmenting UML with decision table technique. In: Stoilov, T., Rachev, B. (Eds.), *Proceedings of the International Conference on Computer Systems and Technologies (CompSysTech'09)*, Rousse, Bulgaria, pp. IIIA.21-1-21-6.

O'Connor, M. (2010). *SWRLLanguage*.
`http://protege.cim3.net/cgi-bin/wiki.pl?SWRLLanguageFAQ` (May 2010).

OMG. (2003a). *UML 2.0 OCL Specification*. OMG document.
`http://www.omg.org/docs/ptc/03-10-14.pdf` (September 2009).

OMG. (2003b). *Unified Modeling Language Specification*. OMG document.
`http://www.omg.org/docs/formal/03-03-01.pdf` (September 2009).

OMG. (2005). *Ontology Definition Metamodel*. OMG document.
`http://www.omg.org/docs/ad/05-08-01.pdf` (September 2009).

OMG. (2009). *Production Rule Representation (PRR)*. OMG document, v. 1.0.
`http://www.omg.org/spec/PRR/1.0/PDF/` (May 2010).

OMG. (2008). *Semantics of Business Vocabulary and Business Rules (SBVR)*. Version 1.0.
`<http://www.omg.org/docs/formal/08-01-02.pdf` (December 2008).

Owre, S., Rushby, J., Shankar, N. (1995). Formal verification for fault-tolerant architectures: prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2), 107–125.

Pardillo, J., Mazon, J.N., Trujillo, J. (2010). Extending OCL for OLAP querying on conceptual multidimensional models of data warehouses. *Information Sciences*, 180(5), 584–601.

Raj, A., Prabhakar, T.V., Hendryx, S. (2008). Transformation of SBVR business design to UML models. In: *Proceedings of the 1st India Software Engineering Conference (ISEC '08)*. ACM, New York, pp. 29–38.

Ross, R.G. (1997). *The Business Rule Book. Classifying, Defining and Modeling Rules. Business Rules Solutions*. LLC, Houston.

Sirin, E., Tao, J. (2009). Towards integrity constraints in OWL. In: Hoekstra, R., Patel-Schneider, P.F. (Eds.), *Proceedings of the 6th International Workshop on OWL: Experiences and Directions (OWLED 2009),* Chantilly, Virginia, USA.
`http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-529/` (March 2010).

Siripornpanit, N., Leckcharoen, S. (2009). An adaptive algorithms translating and back-translating of object constraint language into structure query language. In: *Proceedings of the 2009 International Conference on Information and Multimedia Technology*, pp. 149–151.

Spivey, J.M. (1988). Understanding Z – a specification language and its formal semantics. *Cambridge Tracks in Theoretical Computer Science*, Vol. 3.

Trinkunas, J., Vasilecas, O. (2009). Ontology transformation: from requirements to conceptual model. *Journal of Computer Science and Information Technologies*, 751, 54–68.

Vasilecas, O., Kalibatiene, D., Guizzardi, G. (2009). Towards a formal method for transforming ontology axioms to application domain rules. *Information Technology and Control*, 38(4), 271–282.

von Halle, B. (2002). *Business Rules Applied: Building Better Systems Using the Business Rules Approach*. Wiley, New York.

Wand, Y., Storey, V.C., Weber, R. (1999). An ontological analysis of the relationship construct in conceptual modelling. *ACM TODS*, 24(4), 494–528.

Zacharias, V. (2008). Technical Report: Development and Verification of Rule Based Systems – A Survey of Developers. Technical report.
`http://vzach.de/papers/2008_SurveyTechReport.pdf` (May 2009).

D. Kalibatienė, dr., is full time docent at the Information Systems Department, and researcher at the Information Systems Research Laboratory in Vilnius Gediminas Technical University. She participated in the High Technology Development Program Project "Business Rules Solutions for Information Systems Development (VeTIS)". She is the member of the European Committee and Lithuanian Government supported SOCRATES/ERASMUS Thematic Network project "Doctoral Education in Computing" (ETN DEC) and "Teaching, Research, Innovation in Computing Education" (ETN TRICE). She is author and co-author of more than 30 papers and 1 book in the field of information systems development. Research interests: business rules and ontology based information systems development and conceptual modelling.

O. Vasilecas, prof. dr., is full time professor at the Information Systems Department, principal researcher and head of Information Systems Research Laboratory in Vilnius Gediminas Technical University. He is author and co-author of more than 250 research papers and 5 books in the field of information systems development. His research interests include knowledge, represented by business rule and ontology, based information systems development. He delivered lectures in 7 European universities including London, Barcelona, Athens and Ljubljana. O. Vasilecas carried out an apprenticeship in Germany, Holland, China, and last time in Latvia and Slovenia universities. He supervised 9 successfully defended doctoral theses and now is supervising 5 doctoral students more. He was leader of number of international and local projects. Last time he leaded "Business Rules Solutions for Information Systems Development (VeTIS)" project carried out under High Technology Deelopment Program of Lithuania.

Ontologijos aksiomų taikymas OCL ribojimams gauti iš PAL ribojimų

Diana KALIBATIENĖ, Olegas VASILECAS

Pastaruoju metu informacinių sistemų srityje ontologijos naudojamos tokioms problemoms, kaip duomenų modeliavimas, apsikeitimas duomenimis, heterogeninių duomenų ir modelių integravimas arba pakartotinas programinės įrangos naudojimas, spręsti. Tačiau, nors ir pasiūlyta ontologijomis grindžiamų koncepcinio duomenų modeliavimo metodų, šią dieną nepakankamai dėmesio skiriama ontologijos aksiomomis grindžiamam verslo taisyklių modeliavimui. Šiame straipsnyje autoriai parodo, kaip ontologijos aksiomas galima panaudoti verslo taisyklių įgyvendinimui. Savo pasiūlymą autoriai taiko PAL (Protege Axiom Language), kuri yra KIF (Knowledge Interchange Format) dalis, ribojimų (PAL kalba aprašytų ontologijos aksiomų) transformacijai į OCL (Object Constraint Language) ribojimus, kurie yra UML klasųs dalis. Siūlomas būdas aprašytas formalia Z kalba ir įgyvendintas kaip Axiom2OCL įskiepis, kuris naudojamas transformacijos automatizavimui. Atliktas eksperimentas parodė pasiūlyto metodo veiksmingumą.