SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2010. Vol. 756
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 9–32 P.

Ontology-Based Application for Domain Rules Development

Diana Kalibatiene, Olegas Vasilecas
Information Systems Research Laboratory, Vilnius Gediminas Technical University
Saulėtekio av. 11, Vilnius, LT-10223, Lithuania
diana@isl.vgtu.lt, olegas@isl.vgtu.lt

While there is a great interest in rule-based systems and their development, none of the proposed languages or methods has been accepted as a standard technology yet. Nowadays tools used in process of information systems (IS) development are not extended and adapted enough for modelling and implementation of application domain rules. A particular contingent of researchers proposes using of ontology for development of intelligent IS, since ontology is suitable to represent application domain knowledge. We are challenged in using ontology for the development of application domain rules. In this paper we present a method for ontology axioms transformation to application domain rules and describe how ontology-based development of application domain rules is integrated through IS development life cycle.

Keywords: ontology, axiom, application domain rule, transformation, OCL, PAL.

1 Introduction

Nowadays, ontologies representing application domain knowledge are used for the development of modern information systems (IS). A number of authors believe that the use of such ontologies, transformed and/or translated to IS components, help to 1) reduce the costs of a conceptual modelling [1] and 2) assure the ontological adequacy of the IS [1, 2, 3]; and allow to 3) share and reuse a domain knowledge across heterogeneous software platforms [2, 4], and 4) cognise of an application domain. If the IS is a traditional one, application domain knowledge will be just embedded in the standard components of the IS. If it is going to be an *ontology-driven* (or *ontology-based*) IS, then a separate component – *application domain ontology* – will be developed and included in the IS [1].

In the step of an IS conceptual modelling, researchers are challenged to transform application domain ontology to a conceptual data model, since their conceptualisation of a real world is similar. Both see an application domain in terms of concepts, presenting entities of an application domain, relationships between concepts, properties of concepts and rules (in ontology axioms), presenting constrains of an application domain. While a number of approaches and methods for the transformation of application domain ontology to a conceptual data model have been proposed, like [3, 5, 6, 7, 8] etc, there is lack of a formal theory and methods of ontology components transformation to application domain rules.

In the IS development, there is a great interest in the development of the application domain rules. Authors of [9, 10, 11] etc, organisations, such as the Object Management

Group[1] (OMG), have motivated the application of rules. Methods and languages proposed for the development and implementation of application domain rules are: Unified Modelling Language (UML) [12] with Object Constraint Language (OCL) [13], Demuth et al. method [14], the Ross method [15], CDM RuleFrame [16], Semantic Web Rule Language (SWRL) [17], etc.

However, the results of survey presented in [18] shows that a) a large part of rule-based systems are created without any specific development process, b) almost half of the respondents use an integrated development environment (IDE) (such as the Ontostudio[2], Ilog Rule Studio[3], the Visual Prolog IDE[4] or the SWRL tab [19] of Protégé[5]) that allows to edit, load, debug and run rules. For editing rules the most widely used tools are still textual editors (33%), a simple text editor or a textual rule editor with syntax highlighting (28%) and graphical rule editors (26%); c) verification is dominated by testing (90%) and code review (78%). 74% of respondents do testing with actual data, 50% test with contrived data. Advanced methods of test organisation are used by a minority, with only 31% doing regression testing and 19% doing structural testing with test coverage metrics.

None of the proposed languages or methods have been accepted as a standard technology yet, since they are not suitable for modelling all types of rules, as presented in [20], or limit the opportunity for business people to change them because the verbalization of formal languages is not mature enough. Only a few of them deal with reuse of knowledge acquired in the analysis of a particular application domain and automatic implementation of rules.

Current tools used for the development of IS are not extended and adapted enough for modelling and implementing of rules. For example, in MagicDraw[6] OCL is used for defining constraints. However, there is no suitable interface to facilitate the definition of OCL constraints. User should be familiar with OCL. PowerDesigner[7] is suitable for modelling structural rules (using integrity constraints, foreign keys, domains, checks) only. There is no mechanism for defining and validating of dynamic rules.

In this paper we propose using a domain ontology for the development of application domain rules and describe how ontology-based development of application domain rules is integrated with IS the development life cycle. Therefore, Section 2 overviews the related works according to application domain rules and their implementation and a concept of an ontology, Section 3 presents the comparison of ontology axioms with application domain rules, Section 4 describes ontology-based development of application domain rules in IS life cycle, Section 4 presents the application of the proposed in the previous section method of ontology axioms to information processing rules and its implementation to the Axiom2OCL plug-in, and Section 5 concludes the paper.

[1] http://www.omg.org/
[2] http://www.ontoprise.de/en/home/
[3] http://www.ilog.com/products/businessrules/
[4] http://www.visual-prolog.com/
[5] http://protege.stanford.edu
[6] http://www.magicdraw.com/
[7] http://www.sybase.com/products/modelingdevelopment/powerdesigner

2 Development of Application Domain Rules

An enterprise system can be viewed as a three-layered system: the business systems, the IS and the supporting software [21]. Any enterprise consists of several business systems, e.g. it is doing several businesses. A business system consists of several IS, e.g. an IS is created to support a business system. Finally, software systems are created to support IS. Consequently, an enterprise system is effective only then all layers of this system are integrated properly. Concepts should be mapped rightly from higher-level system to lower level systems and lower level systems should be constrained by rules governing processes in higher level systems. Therefore, the concept of a rule is analysed according these three levels of abstraction: business system, IS, and software.

At the business system level, rules are statements that define or constrain some aspects of a particular business domain in a declarative manner. For example, *a customer could not buy more than her / his credit limit permits*. At the IS level, rules are statements that define information processing rules using a rule-based language, like OCL. Expressions of information processing rules are very precise, e.g. terms used in expressions are taken from the particular data model [22]. For example, the following OCL expression "*context c: Company inv enoughEmployees: c.numberOfEmployees > 50*" constrains the number of employees in the Company that must always exceed 50. At the software system level, rules are statements represented using language of a specific execution environment, like Oracle 10g[8], Microsoft SQL Server 2008[9], ILOG JRules[10], etc.

According to the presented definitions of rules and three levels of abstraction, rules can be expressed in three different forms [10, 23]. They are:

- informal – rules are expressed using natural language;
- semi-formal – rules are expressed using rule templates, decision trees, decision tables or a graphical modelling language, like UML or Object Role Modelling (ORM) [24]. Ideally, this form is the basic from which to generate executable rule code [10]. Unfortunately, there is no standard rule specification language. There are various rule languages proposed as part of other modelling approaches;
- formal – rules are expressed using a particular formal language, like OCL, or a rule execution language, like SQL in relational database management systems (DBMS). Rules expressed in formal way can be processed automatically.

Rules expressed in the natural language are well understandable for business people. However, these expressions are ambiguous and can be interpreted in different ways. Authors of [10, 23] propose using rule templates, to avoid ambiguity. However, they do not present any approach of implementing such rules and how rules expressed by rule templates could be transformed to executable form. Nowadays, majority of methods describe ways of transforming formal rules to executable rules. Therefore, the main question is – which language is the most suitable for the development of the complete and integral set of rules? Unfortunately, there is no standard describing rule acquisition from the application domain, rule modelling by a suitable language and rule implementation in an executable environment.

[8] http://www.oracle.com/technology/software/products/database/index.html
[9] http://www.microsoft.com/sqlserver/2008/en/us/overview.aspx
[10] http://blogs.ilog.com/brmsdocs/2008/06/15/ilog-jrules-6-for-architects-and-developers-2/

According to the implementation perspective, it is proposed to classify application domain rules as follows.

- *Structural rules* (terms, definitions, facts, and integrity constraints). Terms, definitions and facts can be implemented by elements of a conceptual data model, for example an entity in an entity-relationship model or a class in a UML class model. Therefore, terms, definitions, facts can be regarded as concepts in an ontology and not as rules. Integrity constraints can be implemented by integrity constraints, like referential integrity constraints, cardinality constraints, and mandatory constraints, of a conceptual data model and in case of UML models expressed as OCL invariants. At software system level, integrity constraints can be implemented like SQL assertions, checks, and foreign keys.
- *Dynamic rules*, which can be expressed by ECA rules and implemented using language of a specific execution environment, like SQL triggers. A dynamic rule is: 1) a dynamic constraint, which restricts transitions from one state of the application domain to another, 2) a derivation rule, which creates new information from existing information by calculating or logical inference from facts, or 3) a reaction rule, which evaluates a condition and upon finding it true performs a predefined action.

Since implementation of structural rules is defined quite precisely (it can be seen from the precise definitions of integrity constraints in a conceptual data model, like CHECK, DOMAIN, NOT NULL, referential integrity and other constraints), we concentrate our research on the implementation of dynamic rules.

Methods and languages proposed to model and implement application domain rules can be classified according to their drawbacks as follows.

1 *Non-existence of any graphical notation* – languages and methods of this category do not have any graphical notation. OCL and all OCL-based languages and methods, like the method presented in [25], CDM RuleFrame environment [16], have no graphical notation.

Nowadays UML is the most popular for modelling of business and information systems [26]. UML has graphical notation, which gives a wide range of possibilities for representing objects and their static and dynamic relationships. The most appropriate diagram for describing structural rules is the class diagram. OCL 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 formality, expressiveness, preciseness, and unambiguity, OCL does not have any graphical notation and thus does not account for an easily comprehensible language.

Commercial organisations, such as Oracle[8], also present their own methods and languages for rules modelling. In [16] CDM RuleFrame environment is presented, where special OCL subset called RuleSLang is developed to represent rules. Later this representation is used for the automatic enforcement of rules using Oracle technologies. The main drawback of this approach is the lack of a graphical notation (the same as with pure OCL) and the tight coupling with commercial products of one vendor.

2 *Non-explicit implementation* – languages and methods of this category do not deal with a way rules be implemented (automated, semi-automated or manual). It is expected that many rules specified by the proposed language will likely be enforced in an automated way; and in such cases, the semi-formal or formal language or method is proposed. The Ross method [15], rule templates presented

by [10] and OMG proposed "*Semantics of business vocabulary and business rules*" (SBVR) [23] can be referred to this category.

The Ross method [15] proposes specific constructs for each of the rule families together with a big number of accompanying constructs, such as special symbols, invocation values, special interpreters, and special qualifiers. However, a big number of modelling constructs makes the language quite complicated. Moreover, Ross does not define any format for the rule model representation and interchange.

In [10] rules are expressed by *rule templates*, which are combination of rule clauses. A simple *rule clause* is of the form *<term1> <operator> <term2>*. A *term* is a noun or a noun phrase with an agreed-upon definition. These include a concept (for example, a customer), a property of a concept (for example, customer-credit-rating-code), a value (for example, female) and a value set (for example, Mon, Tues, Wed, Thurs, Fri). An *operator* is any operator that makes sense for the particular term type. The subsequent terms and operators will exist only if they make sense. According to [10], structural rules are expressed using single rule clauses. For example, a fact can be presented by the template *<term 1> IS COMPOSED OF <term 2>* (for example, *Window IS COMPOSED OF frames*) and a mandatory constraint can be presented by the template *<term 1> MUST BE IN LIST <a, b, c>* (for example, *Gender MUST BE IN LIST <F, M>*). According to (von Halle 2002), a dynamic rule is a combination of rule clauses. For example, reaction rule or action enabler can be presented as *IF <rule clause> THEN <action>* (for example, *IF ordering data = current data THEN insert new record*).

The OMG in [23] "is focused on SBVR as a vehicle for describing businesses rather than their information systems". The OMG proposes to use logical formulations of rules or logical rules, which provide abstract, language-independent syntax for capturing the semantics of a body of shared meaning. These logical formulations are presented by statements and definitions of structured English. Statements are recognised by being fully expressed using the four font styles. For more details see [23]. However, authors of [23] do not define the basic patterns or templates for rule definitions. They just suggest which keywords should be used in rules and how expressions of application domain rules should look like.

3 *Limited type of rules* – languages and methods of this category is limited on modelling a specific type of rules.

In [14] the templates of rules are presented to generate SQL views and triggers, but the trigger action part is not automatically generated. A method presented in [25] is suitable generating triggers from consistency rules defined using OCL, but the authors limit the usage of method to consistency rules only.

4 *Suitable for rule implementation at the lower levels of abstraction* – languages and methods of this category deals with implementation of rules expressed in a formal way. These methods do not deal with elicitation of rules from the application domain. They use rules already expressed in a particular formal language.

Authors in [27] briefly describe currently used methods for generating relational database schemas, their limitations and drawbacks, and propose a method which advances them by generating full-fledged relational database schemas from a conceptual model. The proposed method consists of metamodel-based and pattern-based transformations.

Principles of creating pattern-based transformations are defined for transformation of OCL expressions to corresponding SQL code.

The particular methodologies were selected in [20] and compared according to the possibility of rule modelling. Authors show that common methods are insufficient or at least inconvenient for a complete and systematic modelling of rules. Some relevant enhancements of these methods are more powerful but still emphasise only certain aspects and types of rules.

In [28] authors present how rules can be managed in enterprises and propose the managing scenario. Future works of authors are detailed design of the rule repository, development of the necessary facilities for extraction of rules from organization's business model and specification of the necessary operations for integration of IS repository with the rule repository.

In [29] authors identify the issues that they think are problematic in the context of rule explicit manipulation and present challenges for future research. The focus was put on five areas: the rule scope, acquisition, specification, implementation, and management. For each of the areas authors pointed out the issues that present obstacles for using rules as an approach to IS development.

2.1 The Main Problems Concerned with Domain Rules Modelling

The process of developing the application domain rules involves two main problems – determining the rules (their elicitation from the application domain) and developing ECA rules (their implementation).

First of all, it is necessary to determine the rules of a domain and ensure that they are appropriate. The process of determining which rules apply to a particular situation often involves an open-ended search through multiple sources: business speech, documents, laws, an application domain ontology, etc [11]. A set of application domain rules can be defined using different approaches. The main of them are analysis of documents and questionnaire of business employees. The consensus from all the domain stakeholders should be obtained on the problem of which the rules and their meaning should be used. It is suggested to use business vocabularies or application domain ontologies to ensure the one meaning of an application domain and its rules. When the application domain changes the rules should be properly adapted to new conditions. Capturing, documenting and retaining the domain rules prevent the loss of knowledge when employees leave an enterprise [30].

After the set of appropriate application domain rules is defined, it is necessary to determine which rules will be implemented in a computerised IS. Not all application domain rules are implemented in a supporting computerised IS. These rules are defined in business' documents. Application domain rules, which are going to be implemented in a computerised IS, can be implemented in different ways: by information processing rules and correspondent executable rules of software, as a part of a program code, using rule engines, etc.

The large amount of works on application domain rules elicitation from a domain and implementation in IS shows that this is an important and relevant topic in IS development. However, the lack of a standard method or a language for application domain rules modelling in IS development means that it is not a straightforward problem.

In this paper we propose using ontology for application domain rule modelling and implementation.

2.2 Ontology and Information Systems

Two main directions of this branch may be defined. One is about developing of application domain ontologies and other is about using ontologies for the development of IS. The first one is analysed in ontology engineering field and is not going to be discussed in this paper.

According to [1], every IS has its own ontology, since it ascribes meaning to the symbols used according to a particular view of the world. N. Guarino [1] distinguishes two orthogonal dimensions in IS: a temporal dimension, concerning whether an ontology is used at *development time* or at *run time*, and a structural dimension, concerning the particular way an ontology can affect the main IS components, like application programs, information resources like databases and/or knowledge bases, and user interfaces.

In this paper, the main attention is placed on the usage of ontology at IS development. One of the major trends in this context is using ontology for conceptual data modelling, since a conceptual data model and an ontology both include concepts, relationships between them and rules (in ontology – axioms).

However, it is typically the case that in ontology-based conceptual data modelling approaches the process of developing domain rules is not defined in a formal manner.

Ontology defines the basic concepts, their definitions and their relationships comprising the vocabulary of an application domain and the axioms for constraining relationships and interpretation of concepts [31]. Some authors, like [32], also distinguish properties from concepts. In the simplest case [1], application domain ontology describes a hierarchy of concepts related by particular relationships (e.g., is-a, part-of, etc). In more sophisticated cases, constraints are added to restrict the values of concepts and relationships, like cardinality constraints, possible length, etc. In the most sophisticated cases, suitable axioms are added in order to express and restrict complex relationships between concepts and to constrain their intended interpretation.

In mathematics [33], an axiom is any starting assumption from which other statements are logically derived. It can be a sentence, a proposition, a statement or a rule that enables the construction of a formal system. Axioms cannot be derived by principles of deduction, because they are starting assumptions.

Following the terminology used in [32] and [34], axioms in ontology can be classified as epistemological, consolidation, and derivation axioms. *Epistemological axioms* are defined to show constraints imposed by the way concepts are structured. These include all axioms which can be directly included by the use of modelling primitives and relations that are used in a structural specification of ontology (e.g., is-a relation, part-of relations, cardinality constraints). An example of epistemological axioms imposed by the most basic form of a part-whole relation is: *if exists x and y and x is a part of y, then y is not a part of x* ($\forall x,y\ partOf(x,y) \rightarrow \neg partOf(y,x)$). *Consolidation axioms* impose constraints that exclude unintended interpretations over the structure of the ontology specification. An example of the consolidation axiom from a software quality ontology presented in [35] is: if a product quality characteristic (*qc*) is decomposed in subcharacteristics (*qc1*), then these subcharacteristics should also be a product quality characteristic ($(\forall qc,qc1)(subqc(qc1,qc) \wedge prodqc(qc) \rightarrow prodqc(qc1))(C1)$). Finally, *derivation axioms* allow new knowledge to be derived from the previously existing knowledge represented in the ontology. Typically, derivation axioms are created in order to derive information which can be used to answer the ontology competence questions. An example of a derivation axiom from [35] states that “if there is not a paradigm to which a quality characteristic *qc* is applicable, than *qc* is paradigm-independent” $((\forall qc)\ \neg(\exists p)(applicability(qc, p) \rightarrow pdgInd(qc))$.

If it is necessary, the fourth type of axioms can be defined in addition. They are *definitional axioms* that define the meaning of concepts in ontology.

According to [36], implementation of axioms in ontology modelling environments is:

- restricted in a framework of a description logics [37] or in some kind of logic language, like Knowledge Interchange Format (KIF) [38] in Protégé ontology [39] and SUMO [40], or
- axiom modelling is completely neglected in WordNet [41], which can be used as a lexical ontology, Protégé ontologies (not all), ontologies presented by [42] and [43], DBpedia [44].

This situation is detrimental to the modelling of large-scale ontologies, because it aggravates engineering and maintenance of large sets of axioms [36].

Authors of [36] propose using of objects and categories to represent axioms. They state that categorisation of axioms allows representing the semantics of axioms, and specifying axioms like objects provides a compact, intuitively accessible representation.

Authors of [45] attempt to reduce the difficulty of writing axioms by identifying groups of axioms that manifest common patterns creating templates that allows users to compose axioms by “filling-in-the-blanks”. The method for collecting the templates is also presented in [45]. This method is implemented in Protégé ontology development and management tool.

E. Sirin and J. Tao [64] inspired of growing usage of OWL analyse the possibilities of defining integrity constraint semantics for OWL axioms. Authors implement the proposal in the prototype using Pellet. Authors show that integrity constraints validation can be reduced to SPARQL (Query Language for RDF) query answering using off-the-shelf reasoning. They state that the obtained results show that the goal of using OWL both as a knowledge representation and constraint language for data validation can be achieved without too much effort.

The analysis of ontology development tools, like Protégé, and ontologies, like SUMO, from the implementation perspective shows that epistemological axioms are implemented by structuring concepts in an ontology; consolidation and derivation axioms are not distinguished and they are implemented using some languages suitable for this purpose, like Protégé Axiom Language (PAL) [46] or Ontology Web Language (OWL) [8]. Some consolidation and definitional axioms are implemented by restricting definition of concepts in a particular ontology.

3 Ontology Axioms in Comparison with Application Domain Rules

Here we present differences between ontology axioms, application domain rules, information processing rules, and executable rules, expressed in the form of *event-condition-action* (ECA) rules. This comparison is necessary to define a correct mapping of ontology axioms to application domain rules.

As stated in Section 2, at the IS level rules are statements that define information processing rules using a rule-based language, like OCL, etc. They are taken from the business system level and implement application domain rules. Information processing rules should be precise and expressed as ECA rules to be implemented by executable rules. Therefore, it is necessary to develop ECA rules, which define when the rule should be applied, what should be checked and what to do after checking.

Application domain ontology axioms belong to a particular application domain. They define admissible states of a domain. In particular cases axioms can have conditions under which defined states should be taken.

Table 1 presents the comparison of rules and ontology axioms.

According to this comparison, the following conclusions could be done.

Since axioms can be formalised together with a domain ontology using a particular language, it is reasonable to use this formalisation to automatically transform the ontology axioms to information processing rules or even to executable rules.

Table 1

Ontology axioms in comparison with rules

Criteria of comparison	**Ontology axiom**	**Application domain rule**	**Information processing rule**	**Executable rules**
Level of abstraction	Application domain	Application domain	Information system	Software system (executable environment)
Level of formality	Formal	Informal	Formal	Formal
Languages used to define	PAL, OWL, logic	Natural language	OCL, RuleML, ORM, rule templates, decision trees, decision tables, etc.	A rule execution language, like SQL in relational DBMS
Event	Holds in all cases	Not defined	Insert, update, delete, select	Insert, update, delete, select
Condition	A predicate or a query over the ontology	Explicit or implicit	A predicate or a query over the data model	A predicate or a query over the data model
Action	No action	Explicit or implicit,	Data modification, application specific procedures, transaction operations	Data modification, application specific procedures, transaction operations
State	Predicate over the ontology	Explicit or implicit	No state	No state
Definition	Defined using ontology concepts	Defined using natural language	Defined using data model terms	Defined using data model terms

Protégé axioms and axioms from [35] and [46] were analysed and it was determined that consolidation and derivation axioms have structure *state* or *condition-state*.

A *state axiom* clearly defines a state in which a domain should be and which can be transformed to the condition of an ECA rule. An action can be understood in two ways:

1 if the condition is satisfied, then the transition from one state of the system to another is admissible;

2 if the condition is not satisfied, then the transition is forbidden.

An example of a state axiom, defined by PAL, is presented as follows. It constrains that the number of pages in a newspaper should not exceed 30. This axiom defines a possible state of a newspaper in a domain, i. e. it defines that for all instances of a class newspaper an attribute number_of_pages should not exceed 30.

```
defrange ?Newspaper :FRAME Newspaper
   forall ?Newspaper
(> (number_of_pages ?Newspaper) 30))
```

A *condition-state axiom* defines an admissible state of a domain under the defined condition. In the sense of the ECA structure, a condition-state axiom can be transformed into an ECA rule in two ways:

1 the condition of an axiom is transformed to the condition of an ECA rule, the state of an axiom is transformed to the action of an ECA rule;

2 the condition-state axiom is transformed to an ECA rule as in the case of a state axiom.

An example of a condition-state axiom, defined by PAL, is presented as follows. It constrains that only finished Content (an article or an advertisement) can be included in a Newspaper. This axiom defines a possible state of a newspaper under the defined condition, i. e. it defines that content (an article or an advertisement) can be included in a newspaper. However, it should satisfy a condition – it should be finished.

```
defrange ?Content :FRAME Content
defrange ?Content-SlotVal :FRAME Content 'published_in'
forall ?Content (forall ?Content-SlotVal
   (=> (not('isFinished' ?Content \"must contain\"))
   (instance-of ?Content-SlotVal Newspaper)))
```

Axioms hold in a domain in all cases. However, computer systems should have information when they apply rules. Therefore, according to the structure of an ECA rule, it is necessary to define important events and link them with corresponding rules during the transformation of ontology axioms to information processing rules or executable rules.

4 Ontology-Based Development of Application Domain Rules and IS Life Cycle

This section presents the method of transforming ontology axioms into information processing/executable rules and its mapping to IS development life cycle.

4.1 Transforming Ontology Axioms into Information Processing/Executable Rules

According to the results obtained in Section 2, Fig. 1 presents the basis for ontology axiom-based modelling of application domain rules. The comparison of ontology axioms and application domain rules shows that a) consolidation axioms can be used to model dynamic constraints and / or reaction rules; b) derivation axioms – derivation rules, c) epistemological axioms – the structuring of entities in a conceptual data model, and d) definitional axioms – definitions of entities.

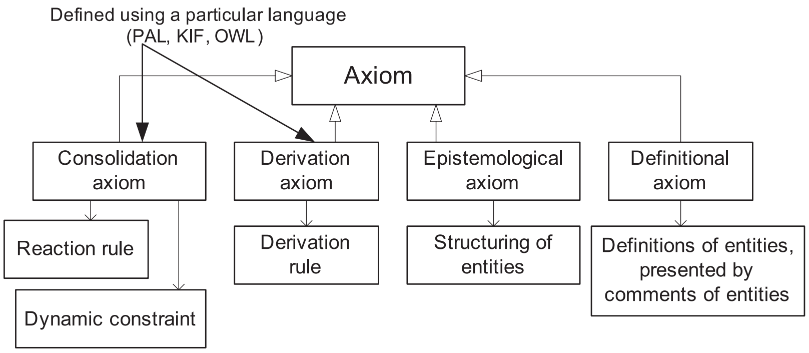

Fig. 1. Ontology axiom-based modelling of application domain rules

The main steps of applying the method of transforming ontology axioms into application domain rules are as follows (Fig. 2).

1 Check if axioms are in an ontology. It warranties that axioms are in an ontology. Otherwise, a user should define axioms.

Note that the creation of an ontology is not analysed here, since, it is not the topic of this paper. The method is based on the assumption that a user of the method has a necessary ontology.

2 Find an axiom.

3 Transform an axiom into a corresponding ECA rule:

3.1 define an event of an ECA rule as insert, update or delete;

3.2 determine the type of the axiom – is it a consolidation or a derivation axiom?

3.2.1 in the case of a consolidation axiom:

note that a consolidation axiom can be a state axiom or a condition-state axiom. However, in the both cases it is transformed to the condition of an ECA rule.

3.2.1.1 Transform an axiom to the correspondent condition of an ECA rule;

3.2.1.2 define an action as (a) if condition is true, then permit the change of a state in a domain, (b) if condition is false, then forbid the change of a state in a domain;

3.2.2 in the case of a derivation axiom:

note that a derivation axiom, which derives new information from the existing information, can be a state axiom or a condition-state axiom.

3.2.2.1 In the case of a state axiom – transform an axiom to the corresponding action of an ECA rule. A condition is always true.

3.2.2.2 In the case of a condition-state axiom: (a) transform a condition to the corresponding condition of an ECA rule, and (b) transform a state to the corresponding action of an ECA rule.

4 End of transformation.

The method is independent of particular languages, which can be used for the definition of axioms and application domain rules.

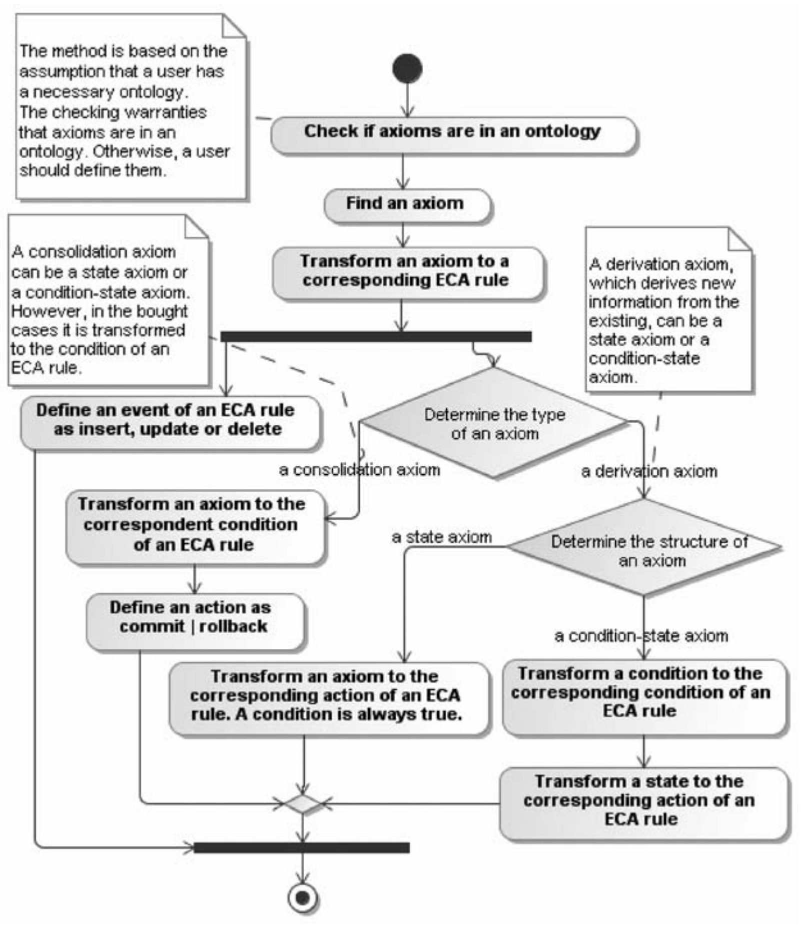
The method is based on the assumption that a user has a necessary ontology. The checking warranties that axioms are in an ontology. Otherwise, a user should define them.
Check if axioms are in an ontology
Find an axiom
Transform an axiom to a corresponding ECA rule
A consolidation axiom can be a state axiom or a condition-state axiom. However, in the bought cases it is transformed to the condition of an ECA rule.
A derivation axiom, which derives new information from the existing, can be a state axiom or a condition-state axiom.
Define an event of an ECA rule as insert, update or delete
Determine the type of an axiom
a consolidation axiom
a derivation axiom
Transform an axiom to the correspondent condition of an ECA rule
a state axiom
Determine the structure of an axiom
Define an action as commit | rollback
a condition-state axiom
Transform an axiom to the corresponding action of an ECA rule. A condition is always true.
Transform a condition to the corresponding condition of an ECA rule
Transform a state to the corresponding action of an ECA rule

Fig. 2. The schema of the proposed method

The formal description of the method is presented in (Vasilecas et al., 2009) and is not discussed here.

4.2 The Mapping of the Proposed Method to IS Development Life Cycle

This sub-section presents the mapping of the proposed method to a system development life cycle. Fig. 3 presents the mapping schema. Business system is presented by application domain ontology, which is created from business documents, laws and various knowledge sources [5]. This ontology with axioms is presented in a formal way, e.g. a particular formal language, like OWL, is used to define the ontology with axioms. Ontology axioms are used to present application domain rules, consolidation axioms are used to model dynamic constraints and/or reaction rules, derivation axioms – derivation rules, epistemological axioms – the structuring of concepts in the ontology, and definitional axioms – definitions of concepts in the ontology.

The ontology with axioms is transformed into the conceptual data model with information processing rules of an IS. The proposed method for transforming ontology axioms into information processing rules is used in this step. The method, described in [5, 8, 47], can be used to transform the ontology into a conceptual data model. However, it is necessary to integrate the obtained conceptual data model and information processing rules. Moreover, if both methods, the method used for the ontology transformation to a conceptual data model and the method used for the ontology axioms transformation to information processing rules, use the same conceptualisation of ontology, then we make an assumption that the obtained conceptual data model and information processing rules will be integral.

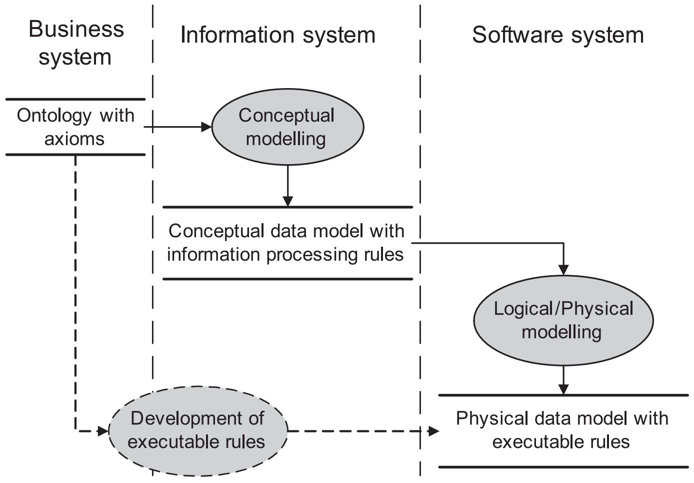
Business system
Information system
Software system
Ontology with axioms
Conceptual modelling
Conceptual data model with information processing rules
Logical/Physical modelling
Development of executable rules
Physical data model with executable rules

Fig. 3. The mapping of the proposed method to the system development life cycle

At the next step, the conceptual data model with information processing rules is transformed into the corresponding physical data model with executable rules. The transformation of a conceptual data model into a physical data model is not analysed here, since there exists a number of tools which support the automatic transformation of a conceptual data model into a physical data model, for example, Sybase PowerDesigner[7], Oracle[8], etc. However, it is important to discuss the possibility of transforming the ontology axioms into executable rules. Since the ontology with axioms is presented in a formal way, the proposed method can be adopted to transform the ontology axioms into executable rules. Such type of the experiment is presented in [31]. However, we believe that the transformation of ontology axioms into information processing rules is more complete and correct, since ontology, in general, is closer to a conceptual data model than to a physical data model, but the transformation of ontology axioms to executable rules is also useful. It helps to facilitate the development of rules and ensure the same conceptualisation of rules at all levels of a system.

The obtained physical data model and executable rules can be implemented in an executable environment.

5 A Case Study of the Transformation of Protégé Axioms into OCL Constraints

This section presents the application of the proposed method for transforming ontology axioms into information processing rules.

5.1 Choosing an Appropriate Ontology Development Tool

First, a suitable ontology development and management tool (ODMT) should be chosen to apply the proposed method.

Currently there are more than 50 different ODMT. A number of authors, such as [48, 49, 50] etc, propose their criteria to assess different ODMT. However, the earlier proposed criteria of ODMT assessment are not enough, since they mainly concentrate on the modelling capabilities of the structure of an ontology and user interfaces. Therefore, according to the perspective of axioms, we select the following criteria, which will be used to analyse existing ODMT.

1 ODMT should support modelling of axioms:
  1.1 ODMT should support an axiom definition language.
  1.2 ODMT should support an axiom management language.
  1.3 ODMT should support syntactical checking of axioms.
2 ODMT availability – ODMT should allow free open source software, which can be installed locally.
3 ODMT usage:
  3.1 ODMT should be user-friendly.
  3.2 ODMT should support graphical notation.
  3.3 ODMT software should be supported by an active project.
4 ODMT should be extensible.

For a detailed study we chose the most popular *WebODE* [51, 52], *OilEd* [53, 54], *Ontolingua* [55, 56], *Protégé, Chimera* [57], *OntoSaurus* [58], *OntoEdit* [59] and *WebOnto* [60, 61] tools. The results of the ODML assessment according to the chosen criteria are presented in Table 2.

According to the results presented in Table 2, the Protégé ontology development and management tool is chosen to support our statement that information processing rules can be elicited from an ontology.

Table 2

Assessment of *WebODE, OilEd, Ontolingua, Protégé, Chimera, OntoSaurus, OntoEdit* and *WebOnto* ontology development and management tools

ODTM / **Criteria**	**WebODE**	**OilEd**	**Ontolingua**	**Protégé**	**Chimera**	**OntoSaurus**	**OntoEdit**	**WebOnto**
Axiom definition and management language (1.1, 1.2)	**Yes (WAB)**	**Yes (DAML + OIL)**	**Yes (KIF)**	**Yes (PAL)**	**Yes (KIF)**	**Yes (KIF)**	**Yes (F Logic)**	**Yes (OCML)**
Checking of axioms (1.3)	**Yes**	**Yes**	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Availability (2)	Free Web Access	[11]	Free Web Access	[11]	[12]	[12]	[13]	Free Web Access
User-friendly (3.1)	**Yes**	No	**Yes**	**Yes**	**Yes**	No	No	**Yes**
Graphical notation (3.2)	[14]	No	[15]	[14]	[14]	No	No	[14]
Active project (3.3)	No	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	No
Extensible (4)	No	No	No	**Plug-ins**	**Plug-ins**	No	**Plug-ins**	No

[11] Open source, installed locally.
[12] Open source and free Web access to evaluation version.
[13] Free Web access to free version. OntoEdit Professional needs software licence.
[14] Graphical taxonomy, graphical view.
[15] Graphical taxonomy, no graphical view.

5.2 Protégé Axiom Language (PAL)

Ontology axioms are implemented in Protégé ontology by the *Protégé Axiom Language* (PAL) constraints [46]. PAL is a superset of the first-order logic, which is used for writing strong logical constraints. PAL can be used to express constraints about a knowledge base, and it can be used to make logical queries about the contents of a knowledge base.

A PAL constraint (or a query) is a statement that holds on a certain number of variables, which range over a particular set of values. Therefore, a constraint or a query in PAL consists of a set of variable range definitions and a logical statement that must hold on those variables. The language of PAL is a limited predicate logic extension of Protégé that supports the definition of such ranges and statements.

The syntax of PAL is a variant of KIF. It supports KIF connectives, but not all KIF constants, predicates (i. e. the theory of arithmetic is much smaller), and statements, like (defrelation) and (deffunction).

PAL provides a set of special-purpose frames to hold the constraints that are added to a Protégé knowledge base, respectively the **:PAL-CONSTRAINT** class. The PAL constraint is an instance of the **:PAL-CONSTRAINT** class. The class has the following slots:

- **:PAL-name**, which holds a label of the constraint;
- **:PAL-documentation**, which holds a natural language description of the constraint;
- **:PAL-range**, which holds the definition of local and global variables that appear in the statement;
- **:PAL-statement**, which holds the sentence of the constraint.

The main part of the PAL constraint is the *PAL-statement,* which can be mapped to the information processing rule. The PAL-statement structure corresponds to the state or condition-state axiom. It has a clearly defined condition and a state. All constraints written by PAL define the state in which the domain should be. However, no information is provided about what should be done to implement a desirable state. The user triggers PAL constraints manually, when it is necessary. For more details about PAL see [46].

The EZPal Tab plug-in [62, 63] is used to facilitate acquisition of PAL constraints without having to understand the language itself. Using a library of templates based on reusable patterns of previously encoded axioms, the interface allows users to compose constraints using a “fill-in-the-blanks” approach.

5.3 Object Constraint Language (OCL)

We use OCL to support our statement that ontology axioms could be transformed into information processing rules, since UML is the most popular language for modelling business and information systems. However, there is no suitable interface to facilitate the definition of OCL constraints. User should be familiar with OCL. We attempt to propose a transformation which simplifies writing OCL constraints.

A UML diagram, such as a class diagram, typically is not refined enough to provide all the relevant aspects of a specification. There is a need to describe additional constraints about the objects in the model. Therefore, OCL has been developed to fill this gap. OCL is a specification language [13]. When an OCL expression is evaluated, it simply returns a value. It cannot change anything in the model. This means that the state of the system will never change because of the evaluation of an OCL expression, even though an OCL expression can be used to specify a state change (e.g., in a post-condition).

According to [13], the following components of OCL constraints are defined:

```
Context TypeName [statement_type] {constrain name}:
[OCL statement]
```

Context introduces the context for OCL constraint. The context can be a particular class, attribute or method of a UML class diagram.

An OCL statement defines an OCL constraint. It is composed of a class, an attribute or a method, which is associated by a mathematical operator with a class, an attribute, a method or a value. An example of a statement is *self.numberOfEmployees > 50*, where *numberOfEmployees* is an attribute and 50 is a possible value of this attribute. *Note* that each OCL constraint is written in the context of an instance of a specific type. In an OCL expression, the reserved word ***self*** is used to refer to the contextual instance. For instance, if the context is the Company class, then self refers to an instance of the Company class.

`[Statement type]` is a possible type of statements in OCL constraints. It defines what kind of statement is used in an OCL constraint. Statement types can be stereotypes (like invariant (**`inv`**), precondition (**`pre`**), and postcondition (**`post`**)), which define stereotypes in an OCL constraint, an initial value (**`init`**), which is used to represent the initial value in an OCL constraint, and derived value (**`derive`**), which is used to represent the derivation rule.

5.4 Mapping PAL with OCL

According to the results presented in the previous section, Table 3 presents mapping of PAL to OCL.

Table 3

Mapping of PAL to OCL

Name of an element	OCL	PAL	Are they mapped?
Name of a constraint (for example, enoughEmployees)	**Yes** Defined after the statement type.	**Yes** Defined after the keyword %3APAL-NAME in quotes.	**Fully mapped**
Description of a constraint	**Yes, but not necessary** Defined in quotes.	**Yes, but not necessary** Defined after the keyword %3APAL-DOCUMENTATION in quotes.	**Fully mapped.** However, it is not the main part of a constraint.

Name of an element	**OCL**	**PAL**	**Are they mapped?**
Type of a constraint	Invariant (`inv`) – associated with a Classifier	**Yes**	**Fully mapped**
	Precondition (`pre`) – associated with an Operation or other behavioural feature	**No**	**No.** PAL has no operations.
	Postcondition (`post`) – associated with an Operation or other behavioural feature	**No**	**No.** PAL has no operations.
	Initial value (`init`) – indicate the initial value of an attribute or association end	**Yes**	**Mapped.** An initial value of an attribute can be defined
	Derived value (`derive`) – indicate the derived value of an attribute or association end	**Yes**	**Mapped.** A derived value of an attribute can be defined
Class, to which a constraint is attached (for example, Organization)	**Yes** Defined after the context.	**Yes.** Defined after the keyword %3APAL-RANGE in quotes.	**Fully mapped**
A statement of a constraint	**Yes** Defined after the statement type or a name, if a name is specified for the constraint. For example, self. numberOfEmployees > 50	**Yes** Defined after the keyword %3APAL-STATEMENT in quotes.	**Fully mapped.** In both constraints classes, attributes and mathematical operators are used.
Condition of a constraint statement	**Yes** Defined after the keyword **if**	**Yes** Defined after the symbol =>	**Fully mapped**
State part of a constraint statement	**Yes** Defined after the keyword **then**	**Yes** Any statement defined after the condition	**Fully mapped**

As can be seen from Table 3, PAL constraints can be transformed into OCL invariants, initial or derived values. Since an ontology and its elements, like classes, has no methods, preconditions and postconditions cannot be presented in an ontology.

An example of the mapping of a PAL statement 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)))
```

- The presented OCL constraint corresponds to the PAL-statement. `?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
```

For more detailed explanations, we present the mapping of Protégé ontology to UML class diagram in Table 4. It is used as a basis to define the mapping of PAL constraints to OCL constraints.

Table 4

Mapping of Protégé ontology elements to UML class diagram elements

Elements of a Protégé ontology	Elements of a 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
class	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

5.5 An Example of Transforming PAL Constraints into OCL Constraints

The prototype of the *Axiom2OCL* plug-in is created to implement the proposed method of transforming PAL constraints into OCL constraints and to support the statement of authors that ontology axioms could be used for the development of information processing rules. The plug-in is developed according to the proposed mapping of PAL to OCL (Table 3).

The plug-in can be attached to MagicDraw UML 15.5 or Protégé 3.0 (or other version). Fig. 4 presents the Axiom2OCL plug-in attached to Protégé 3.4.

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 the 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 will be automatically transformed into OCL constraints.

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

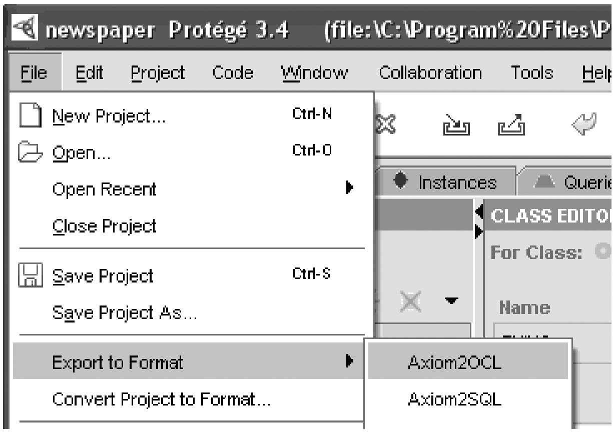

Fig. 4. The *Axiom2OCL* plug-in attached to Protégé 3.4

An example of transforming a PAL constraint, restricting that *the Employee end date should be after the start date,* into the corresponding OCL constraint, 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 ?employee))
\n (> (salary ?editor) (salary ?employee)))))"))
```

- 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. 5), which is generated from the newspaper ontology using UMLBackend plug-in [47].

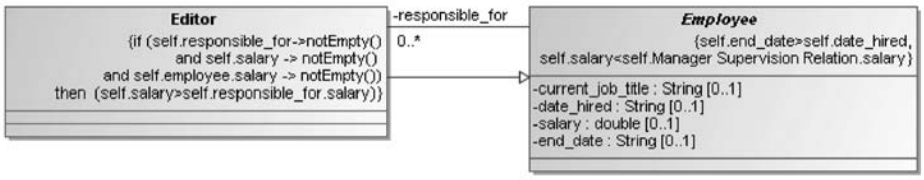
Editor
{if (self.responsible_for->notEmpty()
and self.salary -> notEmpty()
and self.employee.salary -> notEmpty())
then (self.salary>self.responsible_for.salary)}
-responsible_for
0..*
Employee
{self.end_date>self.date_hired,
self.salary<self.Manager Supervision Relation.salary}
-current_job_title : String [0..1]
-date_hired : String [0..1]
-salary : double [0..1]
-end_date : String [0..1]

Fig. 5.Part of a newspaper class diagram

At this moment the prototype is not suitable for transforming all PAL constraints into the corresponding OCL constraints. Therefore, in the future it should be refined and adapted for the transformation of more difficult constraints.

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)"[16] by extending MagicDraw tool to generate OCL constraints from PAL constraints.

6 Conclusions

The analysis of the related works shows that application domain rules are presented in ontology by axioms. However, the majority of authors analyse the use of ontology for the development of a conceptual data model, neglecting or not emphasising ontology axioms as a possible source for business rules development. The comparative analysis of ontology axioms and rules at the level of information systems let us to argue that ontology axioms can be used for modelling rules and consecutive implementation of such rules at the level of software systems.

Syntactic expressions of ontology and a conceptual data model were analysed and it was concluded that consolidation and derivation axioms, expressed in a particular language, can be mapped to dynamic rules, epistemological and definitional axioms – to the structure of a conceptual data model.

Thus, the method for transforming ontology axioms into OCL constraints, which can be defined as a part of a UML class diagram and which are most suitable for representing rules at intermediate level, and next to implementation level should be developed. Such a method is proposed in the paper.

The application of the method for transforming the Protégé ontology axioms (expressed as PAL constraints) to OCL constraints and their implementation in the Axiom2OCL plug-in shows that the method can be used for automatic generation of OCL constraints from ontology axioms.

The next step of the research is extending the developed plug-in.

References

1. N. Guarino. Formal Ontology and Information Systems. In: *Proc. of FOIS'98*. Amsterdam: IOS Press, 1998, pp. 3–15.

[16] http://www.verslotaisykles.lt/VeTIS/

2. M. Jarrar, J. Demey, R. Meersman. On Using Conceptual Data Modeling for Ontology Engineering. In: S. Spaccapietra et al. (eds.), *Journal on Data Semantics*. LNCS, Vol. 2800. Berlin/Heidelberg: Springer, 2003, pp. 185–207.
3. Y. Wand, V. C. Storey, R. Weber. An ontological analysis of the relationship construct in conceptual modeling. *ACM Transactions on Database Systems (TODS)*, Vol. 24(4), 1999, pp. 494–528.
4. T. R. Gruber. Toward Principles for the Design of Ontologies for Knowledge Sharing. *International Journal of Human and Computer Studies*, Vol. 43(4–5), 1995, pp. 907–928.
5. J. Trinkunas, O. Vasilecas. Ontology Transformation: from Requirements to a Conceptual Model. *Acta Universitatis Latviensis [Latvijas Universitates Raksti], Computer Science and Information Technologies*, Vol. 751. University of Latvia, 2009, pp. 54–68.
6. E. Bozsak et al. KAON – Towards a Large Scale Semantic Web. In: K. Bauknecht et al. (eds.), *Proc. of the Third International Conference on E-Commerce and Web Technologies (EC-Web 2002)*. LNCS, Vol. 2455. London: Springer-Verlag, 2002, pp. 304–313.
7. M. A. Goncalves, L. T. Watson, E. A. Fox. Towards a Digital Library Theory: A Formal Digital Library Ontology. *International Journal on Digital Libraries*, Vol. 8(2), 2008, pp. 91–114.
8. OMG: OntologyDefinition Metamodel, 2005. Available: http://www.omg.org/docs/ad/05-08-01.pdf. Accessed September, 2008.
9. T. Morgan. Business Rules and Information Systems: Aligning IT with Business Goals. Boston: Addison-Wesley, 2002.
10. B. von Halle. *Business Rules Applied: Building Better Systems Using the Business Rules Approach*. New York: John Wiley & Sons, 2002.
11. R. G. Ross. *Principles of the Business Rule Approach*. Addison Wesley, 2003.
12. OMG: Unified Modeling Language Specification. Version 1.4.2, ISO/IEC 19501:2005(E) (2005) Available: ftp://ftp.omg. org/pub/docs/formal/05-04-01.pdf. Accessed September, 2008.
13. OMG: UML 2.0 OCL Specification, 2003. Available: http://www.omg.org/docs/ptc/03-10-14.pdf. Accessed September, 2008.
14. B. Demuth, H. Hussmann, S. Loecher. OCL as a Specification Language for Business Rules in Database Applications. In: M. Gogolla, C. Kobryn (eds.), *Proc. of the 4th International Conference on the Unified Modeling Language, Modeling Languages, Concepts, and Tools (UML 2001)*. LNCS, Vol. 2185. London: Springer-Verlag, 2001, pp. 104–117.
15. R. G. Ross. *The Business Rule Book. Classifying, Defining and Modeling Rules*. Houston: Business Rules Solutions Inc., 1997.
16. L. Boyd. CDM RuleFrame – the Business Rule Implementation Framework That Saves You Work. In: *Proc. of ODTUG 2001*. Available: http://www.dulcian.com/odtug_conference.htm. Accessed November, 2006.
17. I. Horrocks, P. F. Patel-Schneider, H. Boley et al. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C document, 2004. Available: http://www.w3.org/Submission/SWRL/. Accessed September, 2009.
18. V. Zacharias. Technical Report: Development and Verification of Rule Based Systems – a Survey of Developers. Technical Report, 2008. Available: http://vzach.de/papers/2008_SurveyTechReport.pdf. Accessed May, 2009.
19. C. Golbreich, A. Imai. Combining SWRL rules and OWL ontologies with Protégé OWL plugin, Jess and Racer. In: *Proc. of the 7th International Protégé Conference*, 2004. Available: http://galimed.med.univ-rennes1.fr/lim/doc_92.pdf. Accessed May, 2009.
20. H. Herbst et al. The Specification of Business Rules: A Comparison of Selected Methodologies. In: A. A. Verrijn-Stuart, T. W. Olle (eds.), *Methods and Associated Tools for the Information System Life Cycle*. New York: Elsevier, 1994, pp. 29–46.
21. A. Caplinskas, A. Lupeikiene, O. Vasilecas. Shared Conceptualisation of Business Systems, Information Systems and Supporting Software. In: H.-M. Haav, A. Kalja (eds.), *Databases and Information Systems* II. The 5th International Baltic Conference «BalticDB&IS'2002», Selected Papers. Dordrecht/Boston/London: Kluwer Academic Publishers, 2002, pp. 109–120.
22. D. C. Hay. *Requirement Analysis. From Business Views to Architecture*. New Jersey: Prentice Hall PTR, 2003.
23. OMG: Semantics of Business Vocabulary and Business Rules (SBVR). Version 1.0, 2008. Available: http://www.omg.org/docs/formal/08-01-02.pdf. Accessed December, 2008.
24. T. Halpin. Object-Role Modeling: an Overview. 1998. Available: http://www.orm.net/pdf/ORMwhitePaper.pdf. Accessed November, 2009.

25. M. Badawy, K. Richta. Deriving Triggers from UML/OCL Specification. In: M. Kirikova (ed.), *Information Systems Development: Advances in Methodologies, Components and Management.* New York: Kluwer Academic/Plenum Publishers, 2002, pp. 305–316.
26. G. Booch, J. Rumbaugh, I. Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, 2000.
27. A. Armonas, L. Nemuraite. Using Attributes and Merging Algorithms for Transforming OCL Expressions to Code. *Information Technology and Control*, Vol. 38(4). Kaunas: Technologija, 2009, pp. 283–293.
28. M. Bajec, M. Krisper. Managing business rules in enterprises. *Electrotechnical Review*, Vol. 68(4), Ljubljana, 2001, pp. 236–241.
29. M. Bajec, M. Krisper. Issues and challenges in business rule-based information systems development. In: D. Bartmann, F. Rajola, J. Kallinkos (eds.), *Proc. of the 13th European Conference on Information Systems (ECIS 2005).* Regensburg: Institute for Management of Information Systems, 2005, pp. 1–12.
30. I. Valatkaite, O. Vasilecas. On Business Rules Approach to the Information Systems Development. In: H. Linger et al. (eds.), *Proc. of the 12th International Conference on Information Systems Development (ISD'2003)*. New York: Springer, 2004, pp. 199–208.
31. O. Vasilecas, D. Kalibatiene, G. Guizzardi. Towards a Formal Method for Transforming Ontology Axioms to Application Domain Rules. *Information Technology and Control*, Vol. 38(4). Kaunas: Technologija, 2009, pp. 271–282.
32. R. A. Falbo, C. S. Menezes, A. R. C. Rocha. A Systematic Approach for Building Ontologies. In: H. Coelho (ed.), *Proc. of the 6th Ibero-American Conference on AI (IBERAMIA'98).* LNAI, Vol. 1484. Berlin Heidelberg: Springer, 1998, pp. 349–360.
33. E. Mendelson. *Introduction to mathematical logic*. Belmont: Wadsworth & Brooks, 1987.
34. G. Guizzardi, R. A. Falbo, J. G. Pereira Filho. Using Objects and Patterns to Implement Domain Ontologies. *Journal of the Brazilian Computer Society*, Special Issue on Software Engineering, Vol. 8(1), 2002. Available: http://www.scielo.br/scielo.php?pid=S0104-65002002000100005&script=sci_arttext&tlng=en. Accessed September, 2008.
35. R. A. Falbo, G. Guizzardi, K. C. Duarte. An Ontological Approach to Domain Engineering. In: *Proc. of International Conference on Software Engineering and Knowledge Engineering (SEKE'02).* New York: ACM, 2002, pp. 351–358.
36. S. Staab, A. Maedche. Ontology Engineering beyond the Modeling of Concepts and Relations. In: N. Guarino et al. (eds.), *Proc. of the ECAI'2000 Workshop on Application of Ontologies and Problem-Solving Methods.* IOS Press, 2000, pp. 15–21.
37. D. McGuinness, P. Patel-Schneider. Usability issues in knowledge representation systems. In: J. Mostow, C. Rich (eds.), *Proc. of AAAI-98*. Madison, Wisconsin: American Association for Artificial Intelligence, 1998, pp. 608–614.
38. M. R. Genesereth. Knowledge Interchange Format (KIF). 2006. Available: http://logic.stanford.edu/kif/kif.html. Accessed October, 2006.
39. N. F. Noy, R. W. Fergerson, M. A. Musen. The knowledge model of Protégé-2000: combining interoperability and flexibility. In: R. Dieng, O. Corby (eds.), *Proc. of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAWÕ00)*. LNAI, Vol. 1937. Berlin: Springer, 2000, pp. 17–32.
40. SUMO: Suggested Upper Merged Ontology (SUMO). 2008. Available: http://www.ontologyportal.org/. Accessed December, 2008.
41. WordNet: Cognitive Science Laboratory. Princeton University, 2006. Available: http://wordnet.princeton.edu/. Accessed December, 2008.
42. R. Culmone, G. Rossi, E. Merelli. An Ontology Similarity Algorithm for BioAgent. In: S. Ercolani, M. A. Zamboni (eds.), *NETTAB Workshop on Agents and Bioinformatics*. Bologna, 2002. Available: http://www.bioagent.net/WWWPublications/Download/NETTAB02P1.pdf. Accessed March, 2007.
43. S. Lin et al. Integrating a Heterogeneous Distributed Data Environment with a Database Specific Ontology. In: E.H.M. Sha (ed.), *Proc. of the International Conference on Parallel and Distributed Systems, 2001 (ICPADS'01)*. Washington: IEEE Computer Society Press, 2001, pp. 430–435.
44. C. Bizer. DBpedia. 2008. Available: http://dbpedia.org/About. Accessed December, 2008.
45. C. S. J. Hou, N. F. Noy, M. A. Musen. A Template-Based Approach toward Acquisition of Logical Sentences. In: M. A. Musen, B. Neumann, R. Studer (eds.), *Proc. of the Conference on Intelligent Information Processing (IIP-2002)*. Montreal: Kluwer, 2002, pp. 77–89.
46. W. Grosso, M. Crubezy. The Protégé Axiom Language and Toolset («PAL»). Stanford Medical Informatics, Stanford University, 2008. Available: http://protegewiki.stanford.edu/index.php/Protege_

Axiom_Language_%28PAL%29_Tabs. Accessed December, 2008.

47. H. Knublauch. UMLBackend. Stanford Medical Informatics, Stanford University, 2007. Available: http://protege.cim3.net/cgi-bin/wiki.pl?UMLBackend. Accessed December, 2008.
48. X. Su, L. Ilebrekke. A Comparative Study of Ontology Languages and Tools. In: A. B. Pidduck et al. (eds.), *Proc. of the 14th International Conference CaiSE.* LNCS, Vol. 2348. London: Springer-Verlag, 2002, pp. 761–765.
49. O. Corcho, M. Fernandez-Lopez, A. Gomez-Perez. Methodologies, tools and languages for building ontologies. Where is their meeting point? *Data & Knowledge Engineering*, Vol. 46(1), 2003, pp. 41–64.
50. L. Casely-Hayford. A comparative analysis of methodologies, tools and languages used for building ontologies. CCLRC Daresbury Laboratories, 2005. Available: http://epubs.cclrc.ac.uk/bitstream/894/OntologyReport.pdf. Accessed June, 2008.
51. J. C. Arpiirez, O. Corcho, M. Fernandez-Lopez, A. Gomez-Perez. WebODE: a scalable ontological engineering workbench. In: Y. Gil, M. Musen, J. Shavlik (eds), *First International Conference on Knowledge Capture.* New York: ACM, 2001, pp. 6–13.
52. A. Gómez-Pérez, M. Fernández-López, O. Corcho. WebODE Ontology Engineering Platform. WebODE Development Group, 2003. Available: http://webode.dia.fi.upm.es/WebODEWeb/index.html. Accessed June, 2008.
53. S. Bechhofer. OilEd Ontology Editor. University of Manchester, 2000. Available: http://xml.coverpages.org/oilEdANn20001204.html. Accessed June, 2008.
54. S. Bechhofer, I. Horrocks, C. Goble, R. Stevens. OilEd: a reasonable ontology editor for the Semantic Web. In: F. Baader, G. Brewka, T. Eiter (eds.), *Proc. of the Joint German/Austrian Conference on Artificial Intelligence (KIÕ01)*. LNAI, Vol. 2174. London: Springer-Verlag, 2001, pp. 396–408.
55. A. Farquhar, R. Fikes, J. Rice. The Ontolingua Server: A Tool for Collaborative Ontology Construction. *International Journal of Human-Computer Studies*, Vol. 46(6), 1997, pp. 707–727.
56. Ontolingua: Ontolingua – Software Description. Stanford University, 2005. Available: http://www.ksl.stanford.edu/software/ontolingua/. Accessed June, 2008.
57. D. L. McGuinness et al. The Chimaera Ontology Environment. In: *Proc. of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence.* AAAI Press/The MIT Press, 2000, pp. 1123–1124.
58. B. Swartout et al. Toward Distributed Use of Large-Scale Ontologies. In: B. Gaines, M. Musen (eds.), *Proc. of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96)*, 1997. Available: http://ksi.cpsc.ucalgary.ca/KAW/KAW96/swartout/Banff_96_final_2.html. Accessed April, 2009.
59. Y. Sure et al. OntoEdit: collaborative ontology engineering for the semantic web. In: *First International Semantic Web Conference (ISWCO02)*. LNCS, Vol. 2342. Berlin: Springer, 2002, pp. 221–235.
60. J. Domingue. Tadzebao and Webonto: Discussing, Browsing and Editing Ontologies on the Web. In: *Proc. of the 11th Knowledge Acquisition Workshop (KAW98),* 1998. Available: http://ksi.cpsc.ucalgary.ca/KAW/KAW98/domingue/. Accessed December, 2008.
61. J. Domingue. WebOnto. 2008. Available: http://kmi.open.ac.uk/projects/webonto/. Accessed June, 2008.
62. C. S. J. Hou, N. F. Noy, M. A. Musen. EZPAL: Environment for composing constraint axioms by instantiating templates. *International Journal of Human and Computer Studies*, Vol. 62(5), 2005, pp. 578–596.
63. C. S. J. Hou. EZPAL. Stanford Medical Informatics, Stanford University, 2008. Available: http://protegewiki.stanford.edu/index.php/EZPal. Accessed December, 2008.
64. E. Sirin, J. Tao. Towards Integrity Constraints in OWL. In: R. Hoekstra, P. F. Patel-Schneider (eds.), *Proc. of OWL: Experiences and Directions 2009 (OWLED 2009),* 2009. Available: http://www.webont.org/owled/2009/papers/owled2009_submission_35.pdf. Accessed March, 2010.