ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2009, Vol. 38, No. 4

TOWARDS A FORMAL METHOD FOR THE TRANSFORMATION OF ONTOLOGY AXIOMS TO APPLICATION DOMAIN RULES[1]

Olegas Vasilecas, Diana Kalibatiene

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

Giancarlo Guizzardi

Ontology and Conceptual Modeling Research Group (NEMO),
Federal University of Espirito Santo, Vitoria – ES – Brazil
e-mail:gguizzardi@inf.ufes.br

Abstract. Ontologies in nowadays are widely used in the process of development of modern information systems (IS), since they are suitable to represent application domain knowledge. However, some aspects of ontology-based IS required to be developed. We propose a formal method for ontology axioms transformation into application domain rules, making them an important and integral part of each application domain and used to constrain or direct different aspects of business. Such rules can be consecutively transformed into an executable form and implemented in a software system of an IS. We propose to use the Z notation for formalisation of previously authors' introduced ontology-based semi-formal method for development of application domain rules.

1. Introduction

In the information systems (IS) development context, researchers use ontology for conceptual data modelling mainly, since a conceptual data model and ontology are closed in some aspects. I.e., both include concepts, relationships between them and rules (in ontology – axioms). The main reasons of applying ontology for IS development are reducing the cost of conceptual analysis, the ontological adequacy of the IS [1, 2, 3], sharing and reusing application domain knowledge across heterogeneous software platforms [1, 4], and cognizing of application domain. However, it is typically the case that in ontology-based conceptual data modelling approaches, a process of developing application domain rules is skipped or not defined in a formal manner in quite rare cases of using.

The importance of rules in IS development process is discussed and motivated by a number of researchers [5, 6, 7], etc. Rules make an important and integral part of each application domain by expressing constraints on concepts, their interpretation, and/or relationships in application domain. A number of methods were proposed to develop rule models: UML with OCL [7, 8, 9], [10], Demuth et al method [11], the Ross method [12], CDM RuleFrame [13] etc. But none of the proposed languages or methods has been accepted as technology standard yet, since they are not suitable for modelling all types of rules [14]. Only a few of them deal with reuse of knowledge acquired in the analysis of some particular application domain and automatic implementation of rules.

We consider that domain ontology should be used in the process of application domain rules modelling for reasons as follows:

a. Domain ontology is about concepts and properties (intrinsic and relational ones) organised in a (taxonomic, mereological) structure, but it is also about excluding unintended interpretations, named as consolidation axioms [15] that can be made on this structure.
b. A widespread design criterion for domain ontologies is the use of competence questions. Derivation axioms are used to answer the set of competence questions by showing that the information necessary to answer these questions is encoded in the ontology.

[1] The work is supported by Lithuanian State Science and Studies Foundation according to High Technology Development Program Project "Business Rules Solutions for Information Systems Development (VeTIS)" Reg. No. B-07042

c. Thus, ontology axioms such as Consolidation and Derivation axioms can be used to model specific types of application domain rules.
d. Finally, as stated in [2], ontology axioms (and ontology as a whole) are typically expressed in a formal way. For this reason, they can in principle be transformed to application domain rules automatically.

We use the Z notation to formalise the approach for mapping ontology axioms to application domain rules previously presented in [16].

The paper is structured as follows. Related works are analysed in Section 2. Our formal characterisation of ontology and conceptual data model using Z are proposed in Section 3. Section 3 also presents the formal transformation rules developed to transform ontology axioms to application domain rules. Section 4 describes the implementation of the proposed method into a developed prototype. Finally, Section 5 concludes the paper.

2. Implementing Rules in IS

Since we address the automatic implementation of application domain rules, they are here analysed at three different abstraction levels.

- At the business system level that can be understood as OMG's MDA computation independent (CIM) level [17], 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 that can be understood as OMG's MDA platform independent (PIM) level [17], rules are statements that define information processing rules using a rule-based language, like OCL [9] etc. Expressions of information processing rules are very precise, e.g. terms used in expressions are taken from the particular data model [18]. For example, the following formal 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 that can be understood as OMG's MDA platform specific (PSM) level [17], rules are statements represented using language of a specific execution environment, like Oracle 10g [19], Microsoft SQL Server 2008 [20], ILOG JRules [21], etc.

Figure 1 presents rules at different abstraction levels.

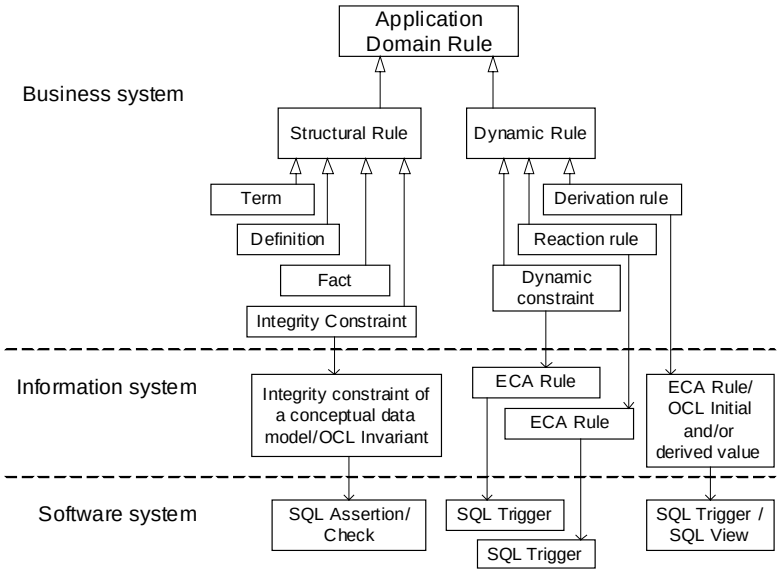

Figure 1. Application domain rules at different abstraction levels[2]

At business system level, application domain rules can be classified into:

- Structural rules (terms, definitions, facts, and integrity constraints), which can be implemented by a conceptual data model of an application domain, e.g., entity-relationship or UML class model (for the sake of simplicity, implementation of terms, definitions and facts is not shown in Figure 1). Therefore, terms, definitions, facts can be regarded as concepts in ontology and not as rules. Integrity constraints can be implemented by conceptual data model integrity constraints, like referential integrity constraints, cardinality constraints, and mandatory constraints, 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.

[2] *Note*: Figure 1 presents implementation of rules only by SQL assertions/checks, triggers and views; however, languages of other execution environments can be used for implementation of rules.

- Dynamic rules, which can be expressed by ECA rules and implemented, like SQL triggers and SQL views (for the case of some derivation rules).
  - A dynamic constraint restricts transitions from one state of the application domain to another.
  - A derivation rule creates new information from existing information by calculating or logical inference from facts.
  - A reaction rule 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. Therefore, the case of domain ontology axioms is analysed in depth.

According to the observation in [2, 22, 23, 24, 25] papers, 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. Some authors, like [25], distinguish properties from concepts also. In the simplest case [2], an 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 field of mathematics [26], 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.

From application domain perspective, axioms are constraints of an application domain, which are in force in all possible situations of interest.

Following the terminology used in [15] and [25], axioms in ontology can be classified in *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 there exists `x` and `y` and `x` is a part of `y`, then `y` is not a part of `x` (`∀x,y partOf(x,y)→ ¬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 [27] is: if a product quality characteristic (*qc*) is decomposed in subcharacteristics (*qc1*), then these subcharacteristics should also be a product quality characteristic (`((∀qc,qc1)(subqc(qc1,qc) ∧ prodqc(qc) → 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 [27] states that "if there is not a paradigm to which a quality characteristic *qc* is applicable, than *qc* is paradigm-independent" (`((∀qc) ¬(∃p)(applicability(qc,p) → 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.

However, the analysis of ontology development tools, like Protégé [28], 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) [29] or OWL [30]. Some consolidation and definitional axioms are implemented by restricting definition of concepts in a particular ontology.

Therefore, the following conclusions, which relate domain ontology axioms and application domain rules, can be drawn:

- Consolidation axioms can be modelled by dynamic constraints and/or reaction rules.
- Derivation axioms can be modelled by derivation rules.
- Epistemological axioms can be modelled by structuring the concepts in a conceptual data model.

Figure 2 presents ontology axiom-based modelling of application domain rules.

Since application domain ontology including axioms can be formalised using some suitable language, like OWL [30], it is reasonable to use this formalisation for automatic transformation of ontology axioms to information processing rules or even to executable rules, like SQL triggers.

The proposed transformation of ontology axioms to application domain rules can be formalised using predicate logic, description logic, denotational semantics, etc.

We use the Z notation [31] in this research to formalise the mapping between ontological axioms and application domain rules that was earlier proposed in [16]. The Z notation is purposed for the formal specification of computer-based systems. It is based on set theory and predicate calculus, and has been

accepted as the ISO standard in 2002 [32]. We have chosen Z, because it is language independent. E.g. we can define mapping of two distinct families of meta-models disregarding languages, which can be used to express those meta-models (like UML, ORM, OWL, etc.). Moreover, the resulting mapping can be implemented by a number of languages, like Java, C++, ATL [33], etc.

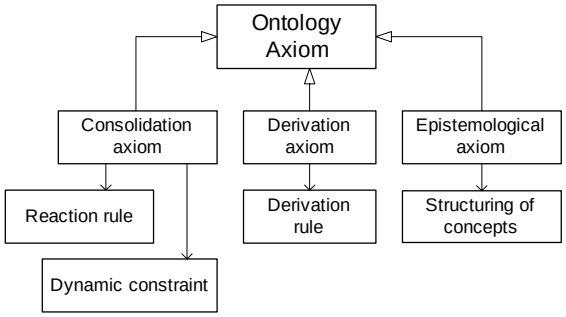

Figure 2. Ontology axiom-based modelling of application domain rules

3. The Method for Transformation of Ontology Axioms to Application Domain Rules Using Z notation

Based on the results of analysis of the related works, we propose the following expression of ontology using Z (Figure 3).

The *Ontology* schema consists of a part above the central dividing line, in which some variables are declared, and a part below the line, which gives a relationship between values of the defined variables, e.g. extra constraints between the defined variables in the form of predicates.

It is important to highlight that this is a syntactic characterisation of an ontology as a specification and one which has been simplified for the specific purposes of this paper. For an in-depth study on issue how ontologies as a specific kind of specification relates to other classes of models such as meta-models, or application-specific conceptual data models (i.e., the real-world semantics of ontologies), one should refer to [34].

Ontology

$concept_i^O : CONCEPT^O \mid i : N \wedge concept_i^O : (P(Z \times CHAR) \mid NOUN)$

$vocabulary_i^O : VOC^O \mid i : N \wedge vocabulary_i^O : P(Z \times CHAR)$

$property_i^O : PROPERTY^O \mid i : N \wedge property_i^O : P(Z \times CHAR)$

$rel_i^O : REL^O \mid i, j : N \wedge rel_i^O : P(Z \times CHAR)$

$value_i^O : VALUE^O \mid i : N$

$defaxiom_i^O : DEFAXIOM^O \mid i : N \wedge defaxiom_i^O : concept_i^O \rightarrow vocabulary_i^O$

$eaxiom_i^O : EAXIOM^O \mid i : N \wedge eaxiom_i^O : (concept_i^O \rightarrow (P(Z \times property_i^O))) \vee (rel_i^O (P(Z \times concept_i^O)))$

$domain_i^O : DOMAIN^O \mid i : N \wedge domain_i^O : (rel_i^O \rightarrow value_i^O) \vee (property_i^O \rightarrow value_i^O)$

$statement_i^O : STAT^O \mid i : N \wedge statement_i^O : (concept_i^O \wedge rel_i^O \wedge (concept_j^O \vee property_i^O))$

$daxiom_i^O : DAXIOM^O \mid i, j : N \wedge daxiom_i^O : (P(Z \times statement_i^O) \rightarrow (concept_j^O \vee statement_i^O \vee (property_i^O \rightarrow value_i^O))$

$caxiom_i^O : CAXIOM^O \mid i : N \wedge caxiom_i^O : P(Z \times statement_i^O) \wedge P(Z \times LC) \wedge P(Z \times QUANT)$

$daxiom_i^O ::= IF[P(statement_i^O)]STATE[P(statement_i^O)]$

$caxiom_i^O ::= IF[P(statement_i^O)]STATE[P(statement_i^O)]$

Figure 3. The expression of *Ontology* using Z

For a formal semantics of many ontological primitives such as different sorts of concept categories or different sorts of mereological relations, one should refer to [35].

Now we would like to introduce main concepts used in the paper. $CONCEPT^O = \{concept^O_i \mid i{:}N\}$ is a set of concepts in an ontology. A *concept* ($concept^O_i$) represents real-world things and is expressed as a word, which is a set of sequences of characters (P ($Z \times CHAR$))[3]. A concept is restricted to be a noun or noun phrase, for example, *customer*. N is a set of positive integers. $VOC^O = \{vocabulary^O_i \mid i{:}N\}$ is a set of definitions. $PROPERTY^O = \{property^O_i \mid i{:}N\}$ is a set of properties in an ontology, where a property ($property^O_i$) represents property types of real-world things, for example, *customer-credit-rating-code*. $REL^O = \{rel^O_i \mid i{:}N\}$ is a set of relationships. A *relationship* (rel^O_i) is a word denoting a relationship between concepts. Some examples of relationships can be *is-a*, *synonym*, *part-of*, *has*, etc. $VALUE^O = \{value^O_i \mid i{:}N\}$ is a set of all possible values in an ontology. For example, value *female* or a set of values *{Mon, Tues, Wed, Thurs, Fri}*. We do not define a type of values. It can be a character, a real number (Q), etc.

$DEFAXIOM^O = \{defaxiom^O_i \mid i : N\}$ is a set of definitional axioms in ontology. A *definitional axiom* ($defaxiom^O_i$) restricts a definition of the particular concept. In general, one concept may have more than one definition. However, in a particular application domain each concept has one particular definition.

[3] (P ($Z \times CHAR$)) expression denotes a set of sequences of characters in all cases in the paper.

$EAXIOM^O = \{eaxiom^O_i \mid i{:}N\}$ is a set of epistemological axioms in ontology. An *epistemological axiom* ($eaxiom^O_i$) associates concepts with a relationship (rel^O_i (P (Z × $concept^O_i$))) or concepts and properties with a relationship ($concept^O_i \rightarrow$ (P (Z × $property^O_i$))). For example, *is-a (bus, vehicle)* means that a bus is a vehicle; *student* → *studentID* means that the concept *student* has the property *studentID*. In general, one concept can be associated with more than one concept and can have a property or a set of properties.

$DOMAIN^O = \{domain^O_i \mid i : N\}$ is a set of domains in an ontology. A *domain* ($domain^O_i$) is a set of possible values ($value^O_i$) of properties ($property^O_i$) and relationships (rel^O_i).

$STAT^O = \{statement^O_i \mid i : N\}$ is a set of statements in an ontology about the domain of interest. They are used to define some types of axioms. A *statement* ($statement^O_i$) is built up of concepts ($concept^O_i$) and relationships (rel^O_i) or concepts ($concept^O_i$), properties ($property^O_i$) and relationships (rel^O_i). An example of a statement is *dolphin is-a mammal*, where there are two concepts (*dolphin* and *mammal*) and one relationship (*is-a*). In some cases, values of properties can be used to aggregate the statement.

$DAXIOM^O = \{daxiom^O_i \mid i : N\}$ is a set of derivation axioms in an ontology. A *derivation axiom* ($daxiom^O_i$) derives new knowledge (at the right-hand side of the arrow) from existing knowledge (at the left-hand side of the arrow).

$CAXIOM^O = \{caxiom^O_i \mid i : N\}$ is a set of consolidation axioms in an ontology. A *consolidation axiom* ($caxiom^O_i$) consists of a combination (set) of statements, which are connected using logical connectives (*LC*) and quantifiers (*QUANT*).

The bottom half of the schema introduces the following extra constraints. Consolidation axioms and derivation axioms are of the form *condition-state*. In the case of a derivation axiom, a condition defines some possible predicate of an application domain, which allows deriving some new state (knowledge) of an application domain. For example, *if customer buys goods for more than 3000 $, it is a gold customer*. Condition (*if-part*) indicates some possible state of the purchasing application domain. When this condition is satisfied, it allows deriving the information about gold customer (new state of customer).

For the completeness of the observation we choose the definition of a conceptual data model from [3] and [36]. Conceptual data modelling (or semantic data modelling) focuses on capturing and representing certain aspects of human perceptions of the real-world so that these aspects can be incorporated into an IS [3]. Most conceptual data modelling approaches are concerned with essential concepts, associations among concepts and constraints of a domain [36].

We propose the following expression of a conceptual data model using Z (Figure 4). Once more, the model presented below is a syntactic definition of a conceptual data model focused on the characteristics which are suitable for the purposes of this paper.

$ENT^{CM} = \{entity^{CM}_i \mid i{:}N\}$ is a set of entities in a conceptual data model. An *entity* ($entity^{CM}_i$) represents types of real-world things, like *person*, *car*, etc. It is restricted to be a noun or noun phrase. *N* is a set of positive integers. $ATTRIB^{CM} = \{attribute^{CM}_i \mid i{:}N\}$ is a set of attributes representing property types of real-world things, like *name*, *age*, etc. $REL^{CM} = \{rel^{CM}_i \mid i{:}N\}$ is a set of relationships, like *is-a*, *part-of*, etc. A *relationship* (rel^{CM}_i) denotes relationships between entities (P (Z × $entity^{CM}_i$) or an entity and a set of attributes ($entity^{CM}_i$ and P (Z × $attribute^{CM}_i$)), e.g. entities are related in a conceptual data model and entities have a particular set of attributes (representing properties). For example, *customer* → *{SSN, first_ name, second_name, birth_data}* means that the entity *customer* has the following attributes *{SSN, first_ name, second_name, birth_data}*; *make (customer, order)* means that the relationship *make* associate two entities *customer* and *order* (in natural language, *customer makes an order*).

$INTEGCONST^{CM} = \{integconst^{CM}_i \mid i : N\}$ is a set of integrity constraints, like *mandatory constraints* (such as *must have*, *must be*, *must be in list*, *must not have*, *must not be*, *must not be in list* or *prerequisite relationship* (for example, *an order must have an order-data*)), *temporal constraints* (for example, *reservation precedes tour* [23]), *mutually-inclusive constraints* (for example, *to travel to a foreign country a VISA is required, based upon citizenship* [23]), *mutually-exclusive constraints* (for example, *a cruise cannot be listed as being sold out and have availability at the same time* [23]), etc.

$VALUE^{CM} = \{value^{CM}_i \mid i{:}N\}$ is a set of values in a conceptual data model. For the sake of simplicity, we do not define a type of values. It can be a character, like *Jone*, a real number (P (Z × *Q*)), like 18, etc. $DOMAIN^{CM} = \{domain^{CM}_i \mid i{:}N\}$ is a set of domains in a conceptual data model. A *domain* ($domain^{CM}_i$) is a set of possible values ($value^{CM}_i$) of attributes ($attribute^{CM}_i$). $CARDCONST^{CM} = \{cardconstraint^{CM}_i \mid i : N\}$ is a set of cardinality constraints of in a conceptual data model. A *cardinality constraint* ($cardconstraint^{CM}_i$) assigns values ($value^{CM}_i$) to relationships (rel^{CM}_i). $RC^{CM} = \{rulecl^{CM}_i \mid i : N\}$ is a set of rule clauses in a conceptual data model. They are used to define dynamic rules. A *rule clause* ($rulecl^{CM}_i$) is of the form (($entity^{CM}_i \vee attribute^{CM}_i \vee value^{CM}_i \vee$ P (Z × $value^{CM}_i$) $\wedge$ $rel^{CM}_i \wedge$ (($entity^{CM}_i \vee attribute^{CM}_i \vee value^{CM}_i \vee$ P (Z × $value^{CM}_i$)) (see (von Halle, 2002)).

$DERRULE^{CM} = \{derrule^{CM}_i \mid i{:}N\}$ is a set of derivation rules in a conceptual data model, which allows to derive new entity, attribute, value or a set of values from the existing rule clause or a set of rule clauses. The rule clause at the left-hand side should be *true* to derive some new information, otherwise it is not applied. $RRULE^{CM} = \{rrule^{CM}_i \mid i{:}N\}$ is a set of reaction rules in a conceptual data model, which test a

condition at the left-hand side and perform an action defined at the right-hand side, which depends on the results of the condition evaluation. The action can be *inserting* new value, *updating* or *deleting* of existing values. $DYNCONST^{CM} = \{dynconst^{CM}_i \mid i{:}N\}$ is a set of dynamic constraints in a conceptual data model, which test a condition at the left-hand side and, if it is true, allow changing of the state of an application domain. Otherwise the changing of the state of an application domain is forbidden. *Note*: integrity constraints ($INTEGCONST^{CM}$), the domain ($DOMAIN^{CM}$) and cardinality constraints ($CARDCONST^{CM}$) in the *Conceptual Data Model* schema belong to structural rules (Figure 1); derivation rules ($DERRULE^{CM}$), reaction rules ($RRULE^{CM}$) and dynamic constraints ($DYNCONST^{CM}$) belong to dynamic rules (Figure 1).

Conceptual Data Model

$$entity_i^{CM} : ENT^{CM} \mid i : N \wedge entity_i^{CM} : (P(Z \times CHAR) \mid NOUN)$$
$$attribute_i^{CM} : ATTRIB^{CM} \mid i : N \wedge attribute_i^{CM} : P(Z \times CHAR)$$
$$rel^{CM}{}_i : REL^{CM} \mid i, j : N \wedge rel_i^{CM} : (P(Z \times CHAR)) \wedge$$
$$rel_i^{CM} : (entity_j^{CM} \rightarrow P(Z \times attribute^{CM}{}_i)) \vee (P(Z \times entity_i^{CM}))$$
$$integconst_i^{CM} : INTEGCONST^{CM} \mid i : N \wedge integconst_i^{CM} : (must \vee must\_not \vee temporal \vee mexc \vee minc) \wedge$$
$$integconst_i^{CM} : (entity_j^{CM} \rightarrow P(Z \times attribute^{CM}{}_i)) \vee (P(Z \times entity_i^{CM}))$$
$$value_i^{CM} : VALUE^{CM} \mid i : N$$
$$domain_i^{CM} : DOMAIN^{CM} \mid i, j, k : N \wedge domain_i^{CM} : (attribute_j^{CM} \rightarrow (P(Z \times value_k^{CM})))$$
$$cardconst_i^{CM} : CARDCONST^{O} \mid i, j, k : N \wedge cardconst_i^{CM} : (rel_j^{CM} \rightarrow value_k^{CM})$$
$$rulecl_i^{CM} : RC^{CM} \mid i : N \wedge rulecl_i^{CM} : (entity_i^{CM} \vee attribute_i^{CM} \vee value_i^{CM} \vee P(Z \times value_i^{CM})) \wedge$$
$$rel^{CM}{}_j \wedge (entity_i^{CM} \vee attribute_i^{CM} \vee value_i^{CM} \vee P(Z \times value_i^{CM}))$$
$$derrule_i^{CM} : DERRULE^{CM} \mid i : N \wedge derrule_i^{CM} : (P(Z \times rulecl_i^{CM}) \mid true \rightarrow$$
$$(entity_i^{CM} \vee attribute_i^{CM} \vee value_i^{CM} \vee P(Z \times value_i^{CM})))$$
$$rrule_i^{CM} : RRULE^{CM} \mid i : N \wedge rrule_i^{CM} : ((P(Z \times rulecl_i^{CM})) \rightarrow do(insert \vee update \vee delete)$$
$$dynconst_i^{CM} : DINCONST^{CM} \mid i : N \wedge dynconst_i^{CM} : ((P(Z \times rulecl_i^{CM}) \mid true) \rightarrow$$
$$do(COMMIT)) \vee ((P(Z \times rulecl_i^{CM}) \mid false) \rightarrow do(ROLLBACK)$$

Figure 4. Expression of a *Conceptual Data Model* using Z

The analysis of the *Ontology* and the *Conceptual Data Model* schemas allows us to state that consolidation axioms can be used to model dynamic constraints or reaction rules, derivation axioms can be used to model derivation rules, definitional axioms can be used to define the meaning of concepts, epistemological axioms can be used to model the structuring of entities in the conceptual data model.

Now we can define the transformation of ontology axioms (consolidation and derivation axioms) to application domain rules (dynamic constraints, derivation and reaction rules) (Figure 5). The schema of transformation contains only the description of transformation of ontology axioms to conceptual data model rules, since the main topic of this paper is ontology axioms and their transformation to application domain rules.

The *Axiom Transformation* schema defines the transformation of derivation axioms to derivation rules and consolidation axioms to reaction rules or dynamic constraints. The transformation of definitional axioms is not presented in the *Axiom Transformation* schema, since definitions of entities are presented in a conceptual data model as comments of entities. The transformation of epistemological axioms is not presented in the *Axiom Transformation* schema, since they are transformed to the structure of a conceptual data model.

AxiomTransformation

$\Xi Ontology$
$\Xi ConceptualDataModel$
$daxiom_i^{O}\,? : DAXIOM^{O}$
$caxiom_i^{O}\,? : CAXIOM^{O}$
$derrule_i^{CM}\,! : DERRULE^{CM}$
$rrule_i^{CM}\,! : RRULE^{CM}$
$dynconst_i^{CM}\,! : DYNCONST^{CM}$

$statement_i^{O}\,? : STAT^{O}$
$rulecl_i^{CM}\,! : RC^{CM}$
$daxiom_i^{O} \rightarrow derrule_i^{CM}$
$caxiom_i^{O} \rightarrow (rrule_i^{CM} \vee dynconst_i^{CM})$

Figure 5. The *Axiom Transformation* schema defined using Z

Since ontology is a source of the transformation, the *Ontology* schema inclusion ($\Xi Ontology$) is used to add all the components of ontology schema to the *Axiom Transformation* schema. Since a conceptual data model is a target of the transformation, the *Conceptual Data Model* schema is also included ($\Xi ConceptualDataModel$) to the same schema. It is used to define the outcome of the transformation. A set of axioms is an input (a variable ending with a question mark (?)) or a source of the transformation. These axioms are transformed into rules of a

conceptual data model, which are output (target) (a variable ending with an exclamation mark (!)) of the transformation.

Extra constraints of the transformation process are the following. Since axioms consist of statements ($statement^{O}_{i}$) and dynamic assertions consist of rule clauses ($rulecl^{CM}_{i}$), statements from ontology axioms should be transformed to the particular rule clauses of a conceptual data model. Examples of aggregation of rules from rule clauses are presented in [6]. For example, the rule "*if customer buys goods for more than 3000 $, it is a gold customer*" can be expressed in the following way: IF *Total_Value* > *3000*, THEN *Customer_Type* = "*gold*". This rule is composed of two rule clauses: "*Total_Value* > *3000*" and "*Customer_Type* = "*gold*"".

The method for the transformation of ontology axioms to application domain rules is defined as follows:

1. Choose an application domain ontology.
2. Check if axioms are in the ontology.

Note that this step warranties that axioms are in the selected ontology. Otherwise, a user should define axioms.

3. Choose an axiom.
4. Transform the axiom to a dynamic constraint, a derivation rule or a reaction rule:
   4.1. determine the type of the selected axiom – is it consolidation or derivation axiom?
   4.2. in the case of a consolidation axiom – transform the consolidation axiom to the corresponding dynamic constraint.

Note that in particular cases a consolidation axiom can be transformed to a reaction rule, when it is not only important to permit or forbid a transition from one state of the application domain to another, but it is necessary to perform a predefined action.

   4.3. in the case of a derivation axiom – transform the derivation axiom to the corresponding derivation rule.
5. End of the transformation.

The transformation of ontology axioms to application domain rules is presented in Figure 5. The application of the proposed method is presented in the next section (Section 4).

4. A Case Study for Transformation of Protégé Ontology Axioms

The ontology for a particular application domain (*Newspaper* [28]), was chosen to illustrate how ontology axioms can be transformed to information processing rules and consequently into executable rules. We have chosen Protégé because it allows to install the open source software locally. A free version of the software provides all features and capabilities required for the present research as well as being user-friendly.

Table 1. Examples of EZPal constraints for the *Newspaper* ontology (CA – consolidation axiom, DA – derivation axiom, EA – epistemological axiom)

No.	Axiom representation in a natural language	EZPal constraint	Classifier
1.	The salary of an editor should be greater than the salary of any employee for which the editor is responsible for.	For every instance I1 of Class *Editor*, if the value of Slot *responsible for*: Class *Editor* has instance I2 of class *Staff*, then Slot salary: Class *Editor* of I1 has a value > to Slot salary: Class *Staff* of I2.	CA
2.	Every advertisement on the same page must be authored by a different salesperson.	Every Instance of Class *Advertisement* that share the same value in Slot *page_number*: Class *Advertisement* must not share values in Slot *salesperson* : Class *Advertisement*.	CA
3.	Author cannot be Editor of the same Article.	For every instance of Class *Article*, Slot *author*: Class *Article* and Slot *editor*: Class *Article* cannot have the same value.	CA
4.	The Newspaper should not include Article, which expiration date (expiration_date) is before (less) then Newspaper's date	For every instance I1 of Class *Article*, if the value of Slot *published_in*: Class *Article* has instance I2 of class *Newspaper*, then Slot *expiration_date*: Class *Article* of I1 has a value less than Slot *date*: Class *Newspaper* of I2.	CA
5.	No two distinct Articles have the same headline.	Every instance of *Article*: Class *Article* has a unique Slot *headline* : Class *Article*.	CA
6.	The new salary of a reporter equals to the 1.1*old salary of a reporter, if he/she writes more then 16 articles per year.	There is no a template for the implementation of this derivation axiom. Therefore, a template base should be extended by a new template for calculating vales of slots from existing values.	DA (mathematical calculation)

The axioms are implemented in Protégé ontology by the Protégé Axiom Language (PAL) constraints [29]. PAL is a superset of the first-order logic, which is used for writing strong logical constraints [29]. The EZPal Tab plug-in [37] 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. Table 1 presents some

examples of EZPal constraints and their classification according to Figure 2.

The detailed analysis of PAL constraints shows that they can be directly transformed to executable rules. Therefore, formal transformation proposed in Section 3 was adopted as follows in this section.

According to [38], we define Protégé ontology using Z in the following way (Figure 6).

$CLASS^{PO} = \{class^{PO}_i \mid i{:}N\}$ is a set of the main concepts in Protégé ontology. A *class* ($class^{PO}_i$) is a real-world thing. $CLASS^{PO}$ of the *Protégé Ontology* schema implements $CONCEPT^{O}$ of the *Ontology* schema. N is a set of positive integers. $SLOT^{PO} = \{slot^{PO}_i \mid i{:}N\}$ is a set of slots presenting properties of classes and their relationships with other classes. $SLOT^{PO}$ of the *Protégé Ontology* schema implements $PROPERTY^{O}$ and a part of REL^{O} (not *is-a*, *inverse* and *has* relationships) of the *Ontology* schema. $REL^{PO} = \{rel^{PO}_i \mid i{:}N\}$ is a set of relationships. They are *is-a*, *inverse* or *has*, where *is-a* is used to present the hierarchical relationship between classes, *inverse* is used to present the inverse relationship between slots, and *has* is used to present slots of a class. REL^{PO} of the *Protégé Ontology* schema implements $EAXIOM^{O}$ of the *Ontology* schema. Some $EAXIOM^{O}$ of the *Ontology* schema can be implemented by some elements (representing relationships between classes) of $SLOT^{PO}$ of the *Protégé Ontology* schema. $VOC^{O} = \{documentation^{PO}_i \mid i{:}N\}$ is a set of class definitions. *Documentation* ($documentation^{PO}_i$) gives a particular description to the class. A class description is a set of sequences of characters (P (Z × *CHAR*)). VOC^{PO} of the *Protégé Ontology* schema implements $DEFAXIOM^{O}$ of the *Ontology* schema. Each class has one particular definition.

$VALUE^{PO} = \{value^{PO}_i \mid i{:}N\}$ is a set of values in an ontology. A value can be *string*, *symbol*, *class*, etc. $VALUE^{PO}$ of the *Protégé Ontology* schema implements $VALUE^{O}$ of the *Ontology* schema.

$DOMAIN^{PO} = \{domain^{PO}_i \mid i : N\}$ is a set of domains in Protégé ontology. A *domain* ($domain^{PO}_i$) is a set of possible values ($value^{O}_i$) of slots ($slot^{PO}_i$). $DOMAIN^{PO}$ of the *Protégé Ontology* schema implements $DOMAIN^{O}$ of the *Ontology* schema.

$STAT^{PO} = \{statement^{PO}_i \mid i{:}N\}$ is a set of statements in Protégé ontology about the domain of interest. They are used to define axioms. A *statement* ($statement^{PO}_i$) is composed of a class ($class^{PO}_i$) or a slot ($slot^{PO}_i$) associated by a relationship (rel^{O}_i) with some class ($class^{PO}_i$) or some slot ($slot^{PO}_i$) or some value ($value^{PO}_i$). $STAT^{PO}$ of the *Protégé Ontology* schema implements $STAT^{O}$ of the *Ontology* schema.

There is not distinguish between derivation and consolidation axioms in Protégé. They are just implemented by PAL constraints. We have defined the PAL constraints of Protégé ontology by Z in a separate schema to simplify and improve the understanding of their definition (Figure 7).

ProtégéOntology

$$class_i^{PO} : CLASS^{PO} \mid i : N \wedge class_i^{PO} : (P(Z \times CHAR)) \mid NOUN$$
$$slot_i^{PO} : SLOT^{PO} \mid i : N \wedge slot_i^{PO} : P(Z \times CHAR)$$
$$rel_i^{PO} : REL^{PO} \mid i, j : N \wedge rel_i^{PO} : ((is-a \vee inverse \vee has) \mid$$
$$((is-a : class_i^{PO} \rightarrow class_j^{PO}) \wedge (inverse : slot_i^{PO} \rightarrow slot_j^{PO}) \wedge (has : class_i^{PO} \rightarrow slot_j^{PO})))$$
$$documentation_i^{PO} : VOC^{PO} \mid i : N \wedge documentation_i^{PO} : (class_i^{PO} \rightarrow P(Z \times CHAR))$$
$$value_i^{PO} : VALUE^{PO} \mid i : N$$
$$domain_i^{PO} : DOMAIN^{PO} \mid i : N \wedge domain_i^{PO} : (slot_i^{PO} \rightarrow value_i^{PO})$$
$$statement_i^{PO} : STAT^{PO} \mid i : N \wedge statement_i^{PO} : ((class_i^{PO} \vee slot_i^{PO}) \wedge rel_i^{PO} \wedge ((class_j^{PO} \vee slot_j^{PO} \vee value_i^{OP})))$$

Figure 6. The *Protégé Ontology* schema defined using Z

PALconstraint

$$\Xi ProtegeOntology$$
$$palname_i^{PO} : PALNAME^{PO} \mid i : N \wedge palname_i^{PO} : P(Z \times CHAR)$$
$$paldocum_i^{PO} : PALDOCUM^{PO} \mid i : N \wedge paldocum_i^{PO} : P(Z \times CHAR)$$
$$palrange_i^{PO} : PALRANGE^{PO} \mid i, j : N \wedge palrange_i^{PO} : P(Z \times class_j^{PO})$$
$$palstatement_i^{PO} : PALSTATEMENT^{PO} \mid i : N \wedge palstatement_i^{PO} : P(Z \times statement_i^{PO}) \wedge P(Z \times LC) \wedge P(Z \times QUANT)$$
$$axiom_i^{PAL} : AXIOM^{PAL} \mid i : N \wedge axiom_i^{PAL} : palname_i^{PO} \wedge paldocum_i^{PO} \wedge palrange_i^{PO} \wedge palstatement_i^{PO}$$

$$palstatement_i^{PO} ::= IF[P(Z \times statement_i^{PO})]STATE[P(Z \times statement_i^{PO})]$$

Figure 7. The *Protégé Ontology Axiom* schema defined using Z

The *PAL constraint* schema defines the Protégé ontology axioms (PAL constraints). The *Protégé Ontology* schema is included (Ξ*ProtegeOntology*) to the *PAL constraint* schema, since its components are used to define PAL constraints. First of all, components of PAL constraints are defined as follows. $PALNAME^{PO} = \{palname^{PO}_i \mid i{:}N\}$ is a set of labels for PAL constraints. Each label is a set of sequences of

characters (P (Z × *CHAR*)). $PALDOCUM^{PO}$ = {$paldocum^{PO}_i$ | *i*:*N*} is a set of a natural language descriptions of PAL constraints. $PALRANGE^{PO}$ = {$palrange^{PO}_i$ | *i*:*N*} is a set of local and global variables that appear in the statement or arrange of the PAL constraint. It is a set of classes ($class^{PO}_i$). $PALSTATEMENT^{PO}$ = {$palstatement^{PO}_i$ | *i*:*N*} is a sentence of the PAL constraint. It is a set of statements ($statement^{PO}_i$), which are connected using logical connectives (*LC*) and/or quantifiers (*QUANT*). $AXIOM^{PAL}$ = {$axiom^{PAL}_i$ | *i*:*N*} is a set of PAL constraints in Protégé ontology. An *axiom* ($axiom^{PAL}_i$) consists of a PAL-name ($palname^{PO}_i$), PAL-documentation ($paldocum^{PO}_i$), PAL-range ($palrange^{PO}_i$) and PAL-statement ($palstatement^{PO}_i$).

An additional constraint is that a PAL-statement ($palstatement^{PO}_i$) is of the form *condition-state*. E.g., a state defines a particular admissible state of an application domain, which can be achieved if a condition is satisfied. Some PAL constraints do not have conditions. They define only admissible state. Moreover, all types of axioms (consolidation and derivation) are defined by PAL constraints in the same manner (see Table 1). Epistemological axioms are implemented by structuring concepts of Protégé ontology.

The definition of SQL triggers was adopted from the *Conceptual Data Model* schema in the following way (Figure 8).

SQLTrigger

$\Xi DataModel$

$\Xi ConceptualDataModel$

$sqltriggercom_i^M : SQLTRIGGERCOM^M \mid i : N \wedge sqltriggercom_i^M : P(Z \times CHAR)$

$sqltriggername_i^M : SQLTRIGGERNAME^M \mid i : N \wedge sqltriggername_i^M : P(Z \times CHAR)$

$sqltable_i^M : SQLTABLE^M \mid i : N \wedge entity_i^{CM} \rightarrow sqltable_i^M$

$sqlattrib_i^M : SQLATTRB^M \mid i : N \wedge attribute_i^{CM} \rightarrow sqlattrib_i^M$

$sqlstat_i^M : SQLSTATMENT^M \mid i : N \wedge$

$sqlstat_i^M : (sqlattrib_i^M \wedge (matop_i^{SQL} \vee is \vee is\_not) \wedge (sqlattrib_i^M \vee value_i^M))$

$sqlevent_i^M : SQLEVENT^M \mid i : N \wedge sqlevent_i^M : (insert \vee update \vee delete \vee select)$

$sqlif_i^M : SQLIF^M \mid i : N \wedge sqlif_i^M : P(Z \times sqlstat_i^M) \wedge P(Z \times LC) \wedge P(Z \times QUANT)$

$sqlthen_i^M : SQLTHEN \mid i : N \wedge sqlthen_i^M : commit \vee rollback \vee insert \vee update \vee delete \vee select$

$sqltrigger_i^M : SQLTRIGGER^M \mid i : N \wedge sqltrigger_i^M : sqltriggercom_i^M \wedge sqltriggername_i^M \wedge$

$sqltable_i^M \wedge sqlevent_i^M \wedge P(Z \times sqlif_i^M) \wedge P(Z \times sqlthen_i^M)$

$sqltrigger_i^M ::= CREATE\ TRIGGER[sqltriggername_i^M][sqltriggercom_i^M]ON[sqltable_i^M][FOR \mid AFTER \mid INSTEAD\ OF]$
$[sqlevent_i^M]AS[FOR\ EACH\ ROW]IF(P(Z \times ([sqlif_i^M][sqlthen_i^M])))$

Figure 8. The *SQL Trigger* schema defined using Z

The *SQL Trigger* schema presents SQL triggers. The *Data Model* schema is included into the *SQL Trigger* schema to represent a particular data model, part of which SQL triggers are. The *Data Model* schema presents a particular physical data model, which is an implementation of a conceptual data model defined by the *Conceptual Data Model* schema. Therefore, the *Conceptual Data Model* schema is also included into the *SQL Trigger* schema. For the sake of simplicity, the *Data Model* schema is not described in details in this paper and is going to be defined in future works.

The elements of SQL trigger schema in Figure 8 are described as follows. $SQLTRIGGERCOM^M$ = {$sqltriggercom^M_i$ | *i*:*N*} is a set of natural language descriptions of SQL triggers. They are sets of sequences of characters (P (Z × *CHAR*)). $SQLTRIGGERNAME^M$ = {$sqltriggername^M_i$ | *i*:*N*} is a set of names of SQL triggers. $SQLTABLE^M$ = {$sqltable^M_i$ | *i*:*N*} is a set relations to which SQL triggers are attached. $sqltable^M_i$ implements a particular entity from the *Conceptual Data Model* schema ($entity^{CM}_i$). $SQLATTRIB^M$ = {$sqlattrib^M_i$ | $i : N$} is a set of attributes, which are used to define SQL triggers. $sqlattrib^M_i$ implements a particular attribute from the *Conceptual Data Model* schema ($attribute^{CM}_i$). $SQLSTAT^{SQL}$ = {$sqlstat^{SQL}_i$ | $i : N$} is a set of statements expressed using SQL. They are used to define SQL triggers. A *statement* ($sqlstat^{SQL}_i$) is composed of attributes ($sqlattrib^M_i$) or an attribute ($sqlattrib^M_i$) and a value ($value^{SQL}_i$), which are associated by a mathematical operator ($matop^{SQL}_i$) or a keyword (*is* or *is not*). An example of a statement is *employee.salary is not null*, where *salary* is an attribute and *null* is a possible value of this attribute. Note that for each attribute in SQL statement the context or table, to which it belongs, is specified before this attribute. In the previous example, the *salary* attribute is taken from the *employee* table.

$SQLEVENT^M$ = {$sqlevent^M_i$ | *i*:*N*} is a set of events activating SQL triggers. They are *insert*, *update*, *delete* or *select*. $SQLIF^M$ = {$sqlif^M_i$ | *i*:*N*} is a set of conditions in a SQL trigger, which have to be checked and evaluated when a SQL trigger is activated. It is a set of statements ($sqlstat^{SQL}_i$), which are connected using logical connectives (*LC*) and/or quantifiers (*QUANT*). $SQLTHEN^M$ = {$sqlthen^M_i$ | *i*:*N*} is a set of actions of a rule, which have to be executed after the condition is evaluated. If SQL trigger implements a dynamic constraint from the *Conceptual Data Model*

schema ($dynconst^{CM}{}_i$), there are two possible actions: (i) if a condition is satisfied, to admit (*commit*) the transition from one state of the system to another, (ii) if a condition is not satisfied, to forbid (*rollback*) the transition from one state of the system to another. If a SQL trigger implements a reaction rule form the *Conceptual Data Model* schema ($rrule^{CM}{}_i$), an action can be *insert*, *update* or *delete*. If a SQL trigger implements a derivation rule from the *Conceptual Data Model* schema ($derrule^{CM}{}_i$), an action is a SQL select statement.

$SQLTRIGGER^M = \{sqltrigger^M{}_i \mid i:N\}$ is a set of SQL triggers. A *SQL trigger* ($sqltrigger^M{}_i$) consists of a comment ($sqltriggercom^M{}_i$), a name ($sqltrigger-name^M{}_i$), a table ($sqltable^M{}_i$), an event ($sqlevent^M{}_i$), a condition ($sqlif^M{}_i$) and an action ($sqlthen^M{}_i$). E.g., it has the structure of ECA rule.

The bottom half of the schema introduces the extra constraint for the *SQL Trigger* schema that a SQL trigger should have the form as presented.

The *PAL SQL Transformation* schema is adapted to the Protégé ontology with PAL constraints and SQL triggers as follows (Figure 9).

The *PAL SQL Transformation* schema defines the transformation of PAL constraints to SQL triggers. Since PAL constraints are the source of the transformation, the *PAL constraint* schema (Ξ*PAL-constraint*) is included to the *PAL SQL Transformation* schema. Since SQL triggers are the target of the transformation, the *SQL Trigger* schema is also included (Ξ*SQLTrigger*) into the same schema. It is then used to define the output of the transformation. A set of PAL constraints ($axiom^{PAL}{}_i$) is input (a variable ending with a question mark (?)) or a source of the transformation. Those PAL constraints are transformed to SQL triggers, which are the output (target) (a variable ending with an exclamation mark (!)) of the transformation.

PALSQLtransformation

$\Xi PALconstra\,int$

$\Xi SQLTrigger$

$axiom_i^{PAL}\,?:AXIOM^{PAL}$

$sqltrigger_i^M\,!:SQLTRIGGER^M$

$sqltriggername_i^M\,!:=palname_i^{PO}\,?$

$sqltriggercom_i^M\,!:=paldocum_i^{PO}\,?$

$sqltable_i^M\,!:=palrange_i^{PO}$

$sqlevent_i^M\,!:=insert\vee update\vee delete\vee select$

$sqlif_i^M\,!:=palstatement_i^{PO}\,?$

$sqlthen_i^M\,!:=commit\vee rollback\vee insert\vee update\vee delete\vee select$

Figure 9. The *PAL SQL Transformation* schema defined using Z

Extra constraints of the transformation are the following. PAL-name ($palname^{PO}{}_i$) should be transformed to SQL trigger name ($sqltriggername^M{}_i$). PAL-documentation ($paldocum^{PO}{}_i$) should be transformed to comments of SQL trigger ($sqltriggercom^M{}_i$). PAL-range ($palrange^{PO}{}_i$) should be transformed to SQL table ($sqltable^M{}_i$). An SQL event is insert, update, delete or select. PAL-statement ($palstatement^{PO}{}_i$) should be transformed to SQL if ($sqlif^M{}_i$). SQL then ($sqlthen^M{}_i$) is commit, rollback, insert, update, delete or select.

A prototype of the *AxiomTransformation* plug-in was developed to carry out the experiment of automatic transformation of ontology PAL constraints to SQL triggers. In this prototype it is necessary to specify a file, where SQL triggers will be stored and all axioms are then automatically transformed to SQL triggers.

At the moment plug-in is suitable for the implementation of dynamic constraints only. Therefore, in the future it should be extended to cover also the transformation of derivation axioms to derivation rules.

Table 2 as example presents results of a transformation of a PAL constraint to the corresponding SQL trigger.

Some corrections should be made to SQL triggers obtained so that they can be implemented in specific ADBMS. The user should choose an activation time (`FOR|AFTER|INSTEAD OF`) and an event of triggering (`[DELETE] [,][INSERT][,][UPDATE]`). The user should also link the generated SQL triggers with particular databases, since some names of attributes or tables may vary depending on implementation details.

The next step of the research is extending the developed prototype. Firstly, it is necessary to extend the prototype for the transformation of all types of axioms that we discussed in this paper; secondly, the proposed schemas of Z should be extended and refined according to the results of observations obtained.

5. Conclusions

The analysis of the related works in the field of knowledge-based information systems development using the domain ontology shows that application domain rules are part of knowledge represented in such ontology. Rules are represented in the ontology by axioms and defined using ontology concepts. However, the topics of using ontology axioms for application domain rules modelling and consequently implementation in software systems are not investigated enough before now.

According to the detailed analysis of ontology axioms and application domain rules, we propose the method for transformation of ontology axioms to application domain rules and define such transformation in formal way using Z notation. The formal definition of the method is based on the syntactic expressions of ontology concepts, conceptual data model and the developed formal rules for transformation of ontology axioms into rules of an application domain. We suggested to use consolidation axioms for modelling of dynamic constraints, derivation axioms – for modelling of derivation rules, epistemological axioms – for

modelling of structuring of concepts, and finally to use definitional axioms for modelling of the concepts meaning.

In order to illustrate the presented approach and prove that it is implementable, transformation rules obtained and defined using Z were applied to transform the Protégé ontology with axioms (PAL constraints) to the rules implemented by SQL triggers.

The prototype of a software system was developed. Using it, the experiment with the automatic transformation of ontology consolidation axioms to SQL triggers (class of dynamic constraints) was carried out. The experiment shows that the suggested method can be implemented and used for the automation of the ontology knowledge transformation into the components of software system of an information system.

Table 2. Transformation of a PAL constraint to a SQL trigger

PAL constraint

```
([newspaper_00000] of %3APAL-CONSTRAINT
(%3APAL-NAME "editor-employees-salary-constraint")
(%3APAL-DESCRIPTION "The salary of an editor should be greater than the salary
of any employee which the editor is responsible for <=> For every instance I1 of
Class Editor, if the value of Slot responsible for : Class Editor has instance I2
of class Staff, then Slot salary: Class Editor of I1 has a value > to Slot salary:
Class Staff of I2.\n\n")
(%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)))))"))
```

SQL trigger

```
/* Documentation */
/* The salary of an editor should be greater than the salary of any employee
which the editor is responsible for <=> For every instance I1 of Class Editor, if
the value of Slot responsible for : Class Editor has instance I2 of class Staff,
then Slot salary: Class Editor of I1 has a value > to Slot salary: Class Staff of
I2. */
CREATE TRIGGER editor-employees-salary-constraint
ON {editor|employee}
{FOR | AFTER | INSTEAD OF}
{[DELETE] [,] [INSERT] [,] [UPDATE]}
AS
FOR EACH ROW
IF (editor.responsible_for=employee.responsible_for AND editor.salary is not
null AND employee.salary is not null) AND (editor.salary>employee.salary)
BEGIN
  COMMIT TRANSACTION
  PRINT ''Transaction is committed.'
END
ELSE
  RAISERROR ('Rule is violated. The salary of an editor should be greater than
the salary of any employee which the editor is responsible for.')
  ROLLBACK TRANSACTION
END
```

References

[1] **M. Jarrar, J. Demey, R. Meersman**. On Using Conceptual Data Modeling for Ontology Engineering. *In S. Spaccapietra et al (eds.), Journal on Data Semantics, Lect. Notes Comput. Sci.* 2800, *Springer*, 2003, 185-207.

[2] **N. Guarino**. Formal Ontology and Information Systems. *Proc. of the International Conference On Formal Ontology In Information Systems (FOIS'98)*. *IOS Press*, 1998, 3-15.

[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, 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, 1995, 907-928.

[5] **Business Rules Group**. Defining Business Rules ~ What Are They Really? 3*rd edn*., 2000 (*September, 2009): http://www.businessrulesgroup.org/first_paper /BRG-whatisBR_3ed.pdf.*

[6] **B. von Halle**. Business Rules Applied: Building Better Systems Using the Business Rules Approach. *John Wiley & Sons*, 2002.

[7] **Business Rules Group**. The Business Motivation Model. Business Governance in a Volatile World. 2005 (*September*, 2009): *http://www.businessrulesgroup.org /second_paper/BRG-BMM.pdf*.

[8] **G. Booch, J. Rumbaugh, I. Jacobson**. The Unified Modeling Language User Guide. *Addison-Wesley*, 2000.

[9] **OMG**. UML 2.0 OCL Specification. 2003 (*September*, 2008): *http://www.omg.org/docs/ptc/03-10-14.pdf*.

[10] **OMG**. Unified Modeling Language Specification. *Version* 1.4.2, *ISO/IEC 19501:2005(E)*, 2005 (*September*, 2008): *ftp://ftp.omg. org/pub/docs/formal /05- 04-01.pdf*.

[11] **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 UML* 2001, *Lect. Notes Comput. Sci.* 2185, *Springer-Verlag*, 2001, 104-117.

[12] **R.G. Ross**. The Business Rule Book. Classifying, Defining and Modeling Rules. *Business Rules Solutions*. *LLC Houston*, 1997.

[13] **L. Boyd**. CDM RuleFrame – The Business Rule Implementation Framework That Saves You Work. *Proc. of ODTUG (Oracle Development Tools User Group) 2001 – Business Rules Simposium*, 2001 (*November*, 2006): *http://www.dulcian.com/odtug_conference.htm*.

[14] **H. Herbst, G. Knolmayer, T. Myrach, M. Schlesinger**. 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*, *Elsevier*, 1994, 29-46.

[15] **G. Guizzardi, R. A. Falbo, J. G. Pereira Filho**. Using Objects and Patterns to Implement Domain Ontologies. *In Proc. of the 15th Brazilian Symposium on Software Engineering*, *Rio de Janeiro, Brazil*, *Journal of the Brazilian Computer Society*, *Special Issue on Software Engineering*, *Vol*.8, *No*.1, 2002 (*September*, 2008): *http://www.scielo.br/scielo.php?pid=S0104-65002002000100005&script=sci_arttext&tlng=en.*

[16] **O. Vasilecas, D. Bugaite**. Ontology-based Information Systems Development: the Problem of Automation of Information Processing Rules. *In E. Neuhold, T. Yakhno (eds.), Proc. of the Fourth International Conference Advances in Information Systems* (*ADVIS*'2006), *Lect. Notes Comput. Sci.* 4243, *Springer*, 2006, 187-196.

[17] **J. Miller, J. Mukerji (eds.)**. MDA Guide Version 1.0.1. *OMG*, 2003.

[**18]** **D. C. Hay**. Requirement Analysis. From Business Views to Architecture. *Prentice Hall PTR*, *New Jersey*, 2003.

[19] Oracle Database Software Downloads. 2008, (*September*, 2008): *http://www.oracle.com/technology/software/products/database/index.html.*

[20] Microsoft SQL Server 2008. 2008, (*September*, 2008): *http://www.microsoft.com/sqlserver/2008/en/us/overview.aspx*.

[21] ILOG JRules. BRMS Resource Center. 2008, (*September*, 2008): *http://blogs.ilog.com/brmsdocs/2008/06/15/ilog-jrules-6-for-architects-and-developers-2/.*

[22] **E. Bozsak, M. Ehrig, S. Handschuh, A. Hotho et al**. KAON – towards a large scale semantic web. *In K. Bauknecht et al (eds.), Proc. of E-Commerce and Web Technologies* 2002, *Lect. Notes Comput. Sci*. 2455, *Springer*, 2002, 304–313.

[23] **V. Sugumaran, V.C. Storey**. The Role of Domain Ontologies in Database Design: An Ontology Management and Conceptual Modeling Environment. *ACM Transactions on Database Systems (TODS)*, *Vol*. 31, *Issue* 3, 2006, 1064–1094.

[24] **Z. Hu, E. Kruse, L. Draws**. Intelligent Binding in the Engineering of Automation Systems Using Ontology and Web Services. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, Vol. 33, *No*.3, 2003, 403-412.

[25] **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 Artificial Intelligence* (*IBERAMIA*'98), *Lisbon, Portugal, Lect. Notes Artif. Int.* 1484, *Springer-Verlag*, 1998, 349-360.

[26] **E. Mendelson**. Introduction to mathematical logic. *Wadsworth & Brooks*, 1987.

[27] **R. A. Falbo, G. Guizzardi, K. C. Duarte**. An Ontological Approach to Domain Engineering. *Proc. of the International Conference on Software Engineering and Knowledge Engineering* (*SEKE*'02), *Italy*, 2002, 351-358.

[28] Protégé. *Stanford Medical Informatics, Stanford University,* 2006 (*November*, 2006): *http://protege.stanford.edu*.

[29] **W. Grosso**. The Protégé Axiom Language and Toolset ("PAL"). *Stanford Medical Informatics, Stanford University*, 2002 (*September*, 2005): *http://protege.stanford.edu/plugins/paltabs/paldocumentation/index.html*.

[30] **OMG**. OntologyDefinition Metamodel. 2005 (*September*, 2008): *http://www.omg.org/docs/ad/05-08-01.pdf*.

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

[32] **ISO**. Information Technology – Z Formal Specification Notation – Syntax. *Type System and Semantics*, *ISO/IEC* 13568:2002, 196.

[33] ATL Documentation. *The Eclipse Foundation*, 2008 (*November*, 2008): *http://www.eclipse.org/m2m/atl/doc/*.

[34] **G. Guizzardi**. On Ontology, ontologies, Conceptualizations, Modeling Languages, and (Meta)Models, Frontiers in Artificial Intelligence and Applications. I*n O. Vasilecas, J. Edler, A. Caplinskas (eds.), Databases and Information Systems* IV*, IOS Press, Amsterdam,* 2007, 18-39.

[35] **G. Guizzardi**. Ontological Foundations for Structural Conceptual Models. *PhD Thesis*, *University of Twente*, *TI-FRS No.* 15, *Universal Press*, *The Netherlands*, 2005.

[36] **G. Guizzardi, H. Herre, G. Wagner**. On the General Ontological Foundations of Conceptual Modeling. In S. Spaccapietra, et al (eds.), *Proc. of* 21*th International Conference on Conceptual Modeling (ER 2002)*, *Lect. Notes Comput. Sci.* 2503, *Springer-Verlag*, *Berlin*, 2002, 65-78.

[37] **J. Hou**. EZPal Tab. *Stanford University*. 2005 (*March*, 2006): *http://protege.stanford.edu/plugins/ezpal/*.

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

Received October 2008.

DOI: 10.5755/j01.itc.38.4.12082