ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2005, Vol. 34, No. 4

REPRESENTATION OF INTEGRITY CONSTRAINTS IN CONCEPTUAL MODELS[1]

Elita Miliauskaitė, Lina Nemuraitė

Kaunas University of Technology, Department of Information Systems
Studentu St. 50-308, LT-51368 Kaunas, Lithuania

Abstract. Integrity constraints are incident part of conceptual models, including part of semantics of problem domain. Analysis of the most important methods of conceptual modelling has revealed that none of them analyze the complete set of integrity constraints needed for making semantically meaningful model. In our previous work the taxonomy of integrity constraints relevant for design of well-formed conceptual models was established. The goal of this paper is to extend capabilities of UML for required types of integrity constraints introducing stereotypes or reusing them from other methods. In contrast with current practice of deferring description of constraints to detailed design, modelling of constraints in the phase of conceptual analysis makes them reusable in various activities: not only in generating DB schema, but also in early verification, validation, transformation to other types of schemas and program code.

Keywords: information systems, integrity constraints, UML, OCL, profile, stereotypes, tags.

1. Indroduction

The raised level of abstraction of presentation of data and precision of modelling are gaining more and more meaning in nowadays. Growing volumes of data and possibilities of automatic dealing with data in e-Business, Data Warehouses, On-Line Analytical Processing, Data Mining and Semantic Web services require for more careful capturing of semantics of problem domain. OMG has defined Business Semantics of Business Rules 0 bringing the capability to express formally the meaning of expressions used in business (business vocabulary and business rules) independently of language. Conceptual model with derivation rules and integrity constraints serves as representation of business terms at the conceptual level of information system, where transformations to logical and physical representations are carried-out for processing and returning results to the business level. These transformations must be two-way and lossless, from business to physical level, as at current rate of business change, hand-written triggers or procedures untraceable from business are not acceptable for majority of organizations. This is today's vision of bridging business and information technologies.

The presentation of information on the logical and physical levels is well defined today in comparison with representation on conceptual and business level. We have accepted 0 UML 0, 0 as the most suitable language for conceptual modelling. The ontological foundations for UML conceptual models are considered in 0, 0, where high-level stereotypes (kinds, subkinds, phases, roles, categories, mixins and role mixins) are proposed for well-founded conceptual modelling. Nevertheless, the precise definition of identities, relationships and other constraints essential for conceptual modelling, is not involved yet in related UML research. In this paper, the possibilities to obtain precise conceptual model are of main importance. Essentially, the problematic elements of conceptual models are integrity constraints that are often unfoundedly deferred to the phase of detail design. Such a situation is typical for existing not standard UML Data Modelling Profiles (e.g. 0), where only fundamental constraints are considered for logical data models on purpose of generation of database schema.

Integrity constraints are incident part of conceptual models intensively used during analysis and design of information systems 0. They are essential for ensuring correctness of information model, and its ability to represent adequate semantics of problem domain. Integrity constraints may be implemented in data storage system or become a part of software code, in order to protect against unallowable changes in data or invoking behaviour that may raise undesirable situations in functioning information system.

[1] The work is supported by Lithuanian State Science and Studies Foundation according to Eureka programme project „IT-Europe" (Reg. No 3473)

In conceptual model, integrity constraint is logical formula, dependent on problem domain, and it must be held true for all meaningful states of information system 0. Conceptual model with integrity constraints is semantically meaningful if constraints are consistent, and it is effective if constraints are not redundant. Inconsistent constraints raise errors and infinite cycles in running software, and redundant constraints worsen operation. Integrity constraints are consistent if system when performing transitions during its life cycle remains in consistent state. Constraints are redundant if they are overlapping or never can be violated. It is not possible to validate absolute consistency of integrity constraints, but it is possible to find and remove some kinds of inconsistencies.

Therefore to create semantically meaningful conceptual model of good quality we should be able to capture required variety of integrity constraints and to validate them. During analysis of the most important methods of conceptual modelling (ER, EER, HERM, ORM, UML, xUML) the types of constraints that are important for well-formed conceptual models, and situations, when these constraints should be used, were established 0. The aim of this paper is to extend UML-related conceptual modelling method for capturing the whole important types of constraints. The design of data and behavioural schemas of various forms (relational, object-relational, object-oriented, XML, or XML-based) of information system must be founded on the same conceptual model, and integrity constraints should be first class entities in it 0, 0.

The UML specification prescribes both a diagram notation and a metamodel. The notation is comparatively complete, capable to document the structural and behavioural characteristics of problem domain and software. However, modelling persistent storage structures has not been the strong point of UML. The UML notation neglects constructs useful for precise analysis, design, development and management of relational schemata. Data modelling-specific notations and techniques have generally been stronger at this task than those oriented around UML. The UML metamodel, however, provides a structure that accommodates semantic information beyond what is typically expressed in the UML notation. UML extension mechanisms (profiles) based on stereotypes, tagged values and constraints provide a basis for expansion of its applicability. Types of integrity constraints established in 0 could be accurately expressed in terms of the UML metamodel and its extensions.

2. Classification of integrity constraints

Integrity constraints categorized according to several criteria were suggested in 0. By constrained element, the integrity constraints were divided into 2 groups: integrity constraints on attribute and groups of attributes (Figure 1) and integrity constraints on relationship and groups of relationships (Figure 2).

The fundamental constraints on attribute or attribute groups like primary identifier, mandatory, uniqueness constraints are used in all the most popular conceptual modelling methods and implemented in DBVS. The possibility to use constraints on relationships depends on what types of relationships are considered in particular method. For relationships, only multiplicity constraints are common to all methods.

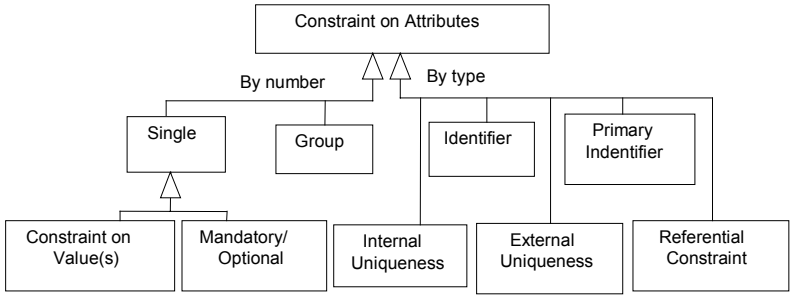

Figure 1. Integrity constraints on attributes

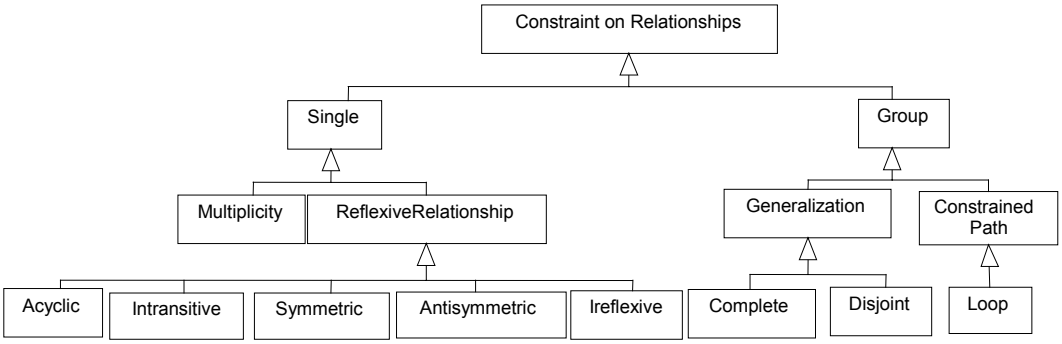

Figure 2. Integrity constraints on attributes and relationship

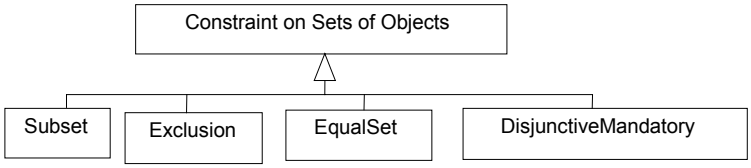

Figure 3. Integrity constraints on sets of objects

By scope for which the constraint is applied integrity constraints where categorized into constraints on single object, constraints on a set of objects, constraints on sets of objects and constraints on sequences of sets of different types of objects. A constraint on sets of objects (Figure 3) consists of set comparison constraints that restrict the way the population of one set of objects compares with that of another compatible set of objects. There are several kinds of set compatible (subset, exclusion, equal set, disjunctive mandatory) constraints that will be defined and analyzed in the following sections.

Looking from perspective of visual modelling of constraints, the ORM 0 – 0 model is quite powerful but also it is rather complex and suitable only to analysis phase. Despite there are methods and capabilities to transform ORM models to (and from) environments of Analysis&Design phase CASE tools (for example, Microsoft Visio-Based Database Modelling in Visual Studio .NET Enterprise Architect), it is difficult to imagine that ORM could be used in practical Information System Engineering projects. Also, it fails to discover cycles and redundancies in long sequences of roles (relationships) comprising loops 0. These constraints are captured in xUML (extended UML) methodology 0. In xUML, the stereotypes are proposed well suited for modelling constraints during development of Information Systems but there are lacking some kinds of constraints on attributes that are practised in EER, ORM and other methods. So it is advisable to unify constraints from different methods to obtain the best expressiveness.

In contrast to ORM, UML is well supported by many CASE tools and widely accepted as standard modelling language, having possibility to describe constraints formally in conceptual language (OCL 0, 0). Because UML is easy extendable, it is possible to extend UML model with stereotypes for visual modelling of whatever constraints that may be defined in other models. It enables transformation of constraints to software code or DBMS functionalities like check functions, triggers or stored procedures. Unfortunately, CASE tools for automatic transformation from OCL to SQL and even OCL parsers are still under development or in early release phases (e.g. 0, the most promising one), so they are still unacceptable for wide use in practical projects.

3. Stereotypes for modelling of integrity constraints

There are alternative options for representation of constraints using UML: natural language, stereotypes and OCL expressions. Stereotypes are useful as patterns not only for discovering constraints, but also for succeeding generation of implementation code, as part of stereotypes directly map to functionality of database, avoiding dealing with complicated expressions. Notation for stereotypes in UML models (part of them are extensions to standard UML) is presented in the next sections, where some of them are illustrated by examples.

UML profiles are lightweight mechanism for its extensions that do not make additions to the UML metamodel, on the contrary to heavyweight extensions requiring for adaptation of model repository interfaces (e.g. JMI interfaces) and interchange formats (e.g. XMI schema's). UML profile is a collection of stereotypes (possibly having tagged values as their attributes) and constraints. Using of profiles is based on the relationships between model elements and stereotypes; it may aim at highlighting the semantics of particular model and/or processing it in a special way. Many standard and non-standard UML profiles are proposed; nevertheless there is a need for precise representation of concepts aiming at development of models of state and behaviour of information systems. In this paper, the main attention is given to modelling that leads to development of data models and schemas of databases, though integrity constraints described as invariants of UML models considered here may evenly be realized by functionality of database as soon as operations of programming language.

A stereotype defines how an existing metaclass may be extended, and enables the use of platform or domain specific terminology or notation in place of or in addition to the ones used for the extended metaclass 0. Just like a class, a stereotype may have properties, which may be referred to as tag definitions. When a stereotype is applied to a model element, the values of the properties may be referred to as tagged values.

Any UML model element can be extended by one or more stereotypes. In conceptual modelling of constraints, stereotypes are applied for constrained elements. Constrained elements of conceptual model may have more than one constraint and, consequently, more than one stereotype. For example, a referential attribute may comprise the part of primary key. Stereotypes may comprise hierarchy. In UML 2.0, it is possible to omit the symbol or label of stereotype in

visual representation and to use tagged values (differently from the earlier versions, tags now cannot exist without stereotypes). When applying stereotypes to constraints, in some cases it is advisable to replace the stereotype with the value of tag, so we omit labels of stereotypes and attach tagged values to constrained elements. The tags and tagged values are concatenated for compact representation. Further, as tagged values are (meta) attributes, they may be typed and their values may be derived on the base of standard or customer defined types of UML model. For example, in 0 the Tag Value Language is proposed as a subset of Perl language for calculating the values of tags. Our work demonstrates how OCL expressions may be used on metamodelling level for definition of profile-related constraints and tagged values in cases when they must be derived from model elements.

4. Integrity constraints on attribute or attributes

Primary identifier is used for unique identification of instance of class (object of object type) among set of all objects of the same type 0, 0, 0. It requires that identifying attribute or attributes group always would have values and that values would be unique. UML does not have special graphical notation for capturing primary identifier constraints, because in object-oriented methodology it is assumed that every class is supported with object identifier (oid), e.g. a memory address assigned by system 0. However, we need visible, attribute-based identifier, defining existence of individuals in problem domain, and these identifiers must be presented in conceptual model. Alternative identifier or oid could be introduced in implementation phase, but nevertheless creation of new objects must be restricted by primary identifier constraint. The primary identifier should be used in conceptual model to capture all the conceptual semantics about a class. To do this we need to introduce extensions to the UML notation. Halpin 0, 0 suggested marking identifying attributes with constraint {P} (Figure 4). The primary identifier in conceptual model does not force to use these attributes as primary key in database, but it sets business requirement that primary identification (and, consequently, uniqueness) of object might be based on this property. If primary identifier was not defined, by default it would be artificial identifier (oid) generated by system (Figure 4). UML provides the Object Constraint Language (OCL) for definition of constraints. OCL expression for constraint of primary identifier comprised of attributes `c1,…,cn` for object type `A` :

```
context A
 inv: self → exists
 (a1,a2:A|a1.c1=a2.c1,…,a1.cn=a2.cn
  implies a1=a2)
```

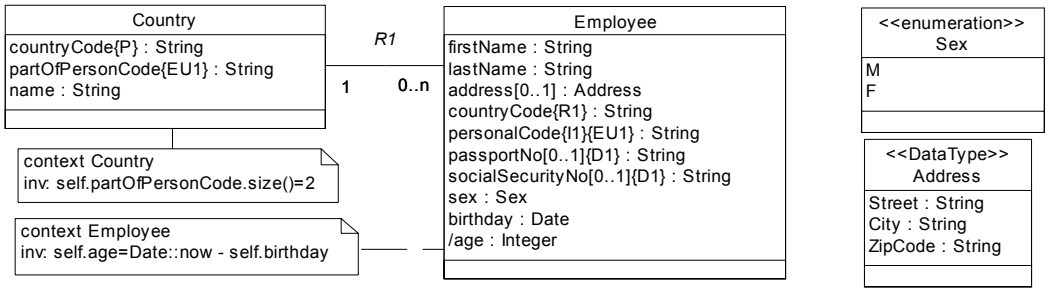

Figure 4. Example of notation of integrity constraints on attributes in UML class diagram

Every object must have one primary identifier, for alternative identification identifier constraint is used. The object may have several identifiers; each of them may consist of several identifying attributes, and identifying attribute may be part of more than one identifier. In UML class diagram the tagged value {In} is used for every identifying attribute (group of attributes) to denote an identifier. For example class Employee besides primary identifier has and identifying attribute personalCode (Figure 4). For alternative identifier constraint, OCL expression is the same as for primary identifier because these constraints have the same meaning but different purpose.

Referential constraint on attribute (group of attributes) identifies that attribute is the identifier of one or more associated objects 0, 0. 0 suggested using the tag for the referential attribute constraint with {Rn}, where Rn is the name of the corresponding association. For example, the attribute Employee.countryCode refers to the primary identifier of Country through the association named R1 (Figure 4). OCL expression for referential constraint of object type `A` that has mandatory relationship with object type `B` (and referential attribute c1 referencing to primary identifier of B – d1):

```
context A
inv: self.B → exists(b:B|b.A=self and
 self.c1=b.d1 and b.d1.oclIsKindOf
  (PrimaryIdentifier))
```

Mandatory constraint on attribute is used to indicate that attribute must have value 0, 0, 0. In UML all class attributes are mandatory by default. Appending [0..1] after attribute name means that attribute is optional 0. In Figure 4, class Employee has optional attributes: passportNo, socialSecurityNo and Address, the rest attributes are mandatory. Mandatory constraint

on attribute can be expressed graphically therefore we don't need OCL expressions.

Disjunctive mandatory constraint indicates that disjunction of class attributes is mandatory. It means that in all allowable states of information system at least one of class attributes constrained by disjunctive mandatory constraint must have value 0, 0, 0. UML does not have graphical notation for disjunctive mandatory constraint so this kind of constraint needs to be expressed textually in an attached note 0, 0 or in OCL. OCL expression for disjunctive mandatory constraint on attributes `a1,…,ai,…,an` of object type `A`:

```
context A
 inv:not(self.a1 isUndefined()) … or
 not(self.ai isUndefined()… or
 not(self.an isUndefined())
```

In order to capture this constraint graphically we need to extend UML notation adding tagged value like {Dn} where in the case of class having several attribute groups constrained with disjunctive mandatory constraint, index n indicates the number of attribute group constrained by this type of constraint. For example, in Figure 4 class Employee has one attribute group {passportNo and socialSecurityNo} constrained by disjunctive mandatory constraint (the number equal to one may be omitted).

Internal uniqueness constraint indicates that the value of one or more attributes of any two instances of class is different from the set of all other instances of that class 0, 0, 0. UML does not have special graphical notation for uniqueness constraint. Hainaut 0 suggested to bold unique attributes, but such solution is not appropriate for unique attribute groups, because one class could have several groups. Halpin 0 has introduced {Un} notation to append as textual constraints to the constrained attributes. The index indicates a group of attributes having unique values for all instances of class. Figure 4 illustrates example of unique attribute for class Country. OCL expression for uniqueness constraint on attributes `c1,…,cn` of object type `A`:

```
context A
 inv: self → exists
  (a1,a2:A|a1.c1=a2.c1,…,a1.cn=a2.cn
   implies a1=a2)
```

External uniqueness constraint could be informally defined like internal uniqueness constraint on attributes of external object type 0, 0 0. The values of attribute (or conceptual join of constrained attribute values) must be different for every instance of class. Figure 4 presents UML class diagram with suggested UML notation extension for graphical representation of external uniqueness constraint, where tag {EUn} marks external attributes that must be unique: Employee.personalCode is alternative identifier defined using external uniqueness constraint EU1 that means pairs Country.partOfPersonalCode and Employee.personalCode have unique value Constraints on Employee. In general case, external uniqueness constraint on attribute `c1` of object `A` and attribute `c2` of related object `B`:

```
context A
inv: self → exists (a1,a2:A|
 a1.B.c1=a2.B.c1 and a1.c2=a2.c2 and
  c1=c2 and c1.oclIsKindOf
   (ExternalUniqueness) implies a1=a2)
```

In UML, the type of attribute constrains its value (for example, in Figure 4 Employee.address has user defined type Address). Value constraint restricts the value of attribute to a finite set of values specified either explicitly (by enumeration), by start and end values (range), or some combination of both. Enumeration types may be modelled as classes, stereotyped as enumeration, with their values listed as attributes 0, 0 (Figure 4). Then type of attribute defined by class with stereotype <<enumeration>> sets constraint on its value (for example, Employee.sex in Fig. 4 may have value "M" or "F"). OCL expressions may specify ranges or other value constraints.

Derived attribute is an attribute whose value can be computed from other attributes already in the model. Such attributes are redundant and generally they are not included in the model. However, there are situations when a clear understanding of a domain is best served by capturing the dependency between attributes explicitly. In UML the derived attribute is denoted by tagged value '/' before derived attribute and the derivation rule is added in the note. In Figure 4 this rule is specified in OCL:

```
context Employee
 inv: self.age = (now -
  self.birthday).toInteger()
```

5. Integrity constraints on relationship or relationships

Multiplicity constraints on relationships are used in all analyzed methods. They define the number of class instances that participate in a relationship. There are two multiplicity indicators for every relationship – one at each end of the line. Sometimes "*" is used as abbreviation of "0..*" meaning "zero or more"; "1" as abbreviation of "1..1" meaning "exactly one"; and "0..1" means "at most one". In general, a multiplicity constraint can be written in the form $[i...j]$, where $0 \le i < N$, $1 \le j \le N$, $i \le j$, and symbol N stands for infinity. Therefore the number of relationships in which an object participates in this role must be, for any instance, between i and j. Role names ra and rb in the diagram (Fig. 5) implies that the role name identifiers can be used as operations ra: B → Set(A) and rb: A → Set(B).

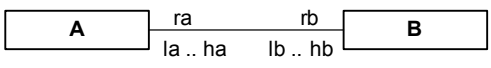

Figure 5. General case of multiplicity constraints on association

For example, ra returns for an object of type A the set of B objects related to the argument. The constraint requires that, for a single A object, the size of the set of related B objects is restricted by the lower bound lb and by the upper bound hb (analogously, for the other side of the association).

```
context A
inv: self.rb → size>=lb
 and self.rb → size<=hb)
context B
inv: self.ra → size>=la
 and self.ra → size<=ha)
```

Generalization/specialization relationship may have several types of constraints 0, but only few of them must be implemented in information system. <<Complete>> constraint on specialization/generalization relationship means that instances of subtypes include all instances of super type. Disjoint constraint means that sets of instances of subtypes do not overlap. In UML, constraint tags may be added in braces beside lines connecting the relevant subtypes, as shown in Figure 6. The following four keywords are predefined in UML for this purpose: "overlapping" (the subtypes overlap), "disjoint" (the subtypes are mutually exclusive), "complete" (all subtypes have been declared), and "incomplete" (some more subtypes may be introduced later). If generalization relationship is not constrained, then subtypes may overlap and do not include all instances of super type. "Complete" constraint on generalization/specialization relationship connecting two subclasses `B` and `C` with super class `A` can be expressed using the following expression:

```
context A
inv: self.B → exists(b|b.A=self
 or C → exists(c|c.A=self))
```

The following OCL expression captures disjoint constraint between subclasses `B` and `C` of super class `A`:

```
context A
inv: B → forAll(b:B|
 C → forAll(c:C|b.self<>c.self))
```

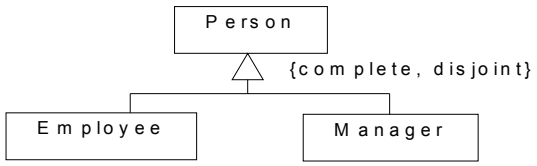

Figure 6. Notation for integrity constraints on generalization relationship

Reflexive associations often have constraints. There are six types of constraints for reflexive relationship analyzed in 0 (where they are named as "ring constraints"):

"Irreflexive" constraint prevents an instance from participating on both sides of a relationship at the same time, i.e., any instance of class A cannot have association with itself. "Asymmetric" constraint prevents the inverse of the relationship between two different instances, i.e. an instance of A has association with instance of B and inverse relationship cannot exist. "Acyclic" constraint prevents a "cycle" such that an instance cannot be the parent, grandparent etc of itself. An acyclic constraint implies asymmetric and irreflexive. "Intransitive" means nobody is a parent of any of his/her grandchildren, i.e. alternative links between instances cannot exist. "Symmetric" constraint requires the existence of inverse association. "Antisymmetric" constraint prevents from inverse relationship as "asymmetric" one, but differently from it allows the same instance to participate in both constraint roles.

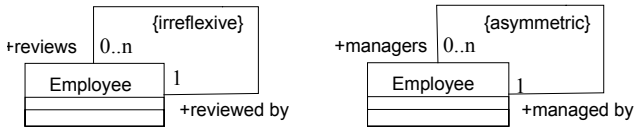

Figure 7. Notation for irreflexive and asymmetric constraints on reflexive associations

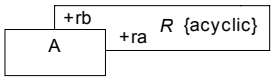

Figure 8. Reflexive relationship with acyclic constraint

UML does not provide built-in constraints for reflexive associations, except possibility to specify them as textual constraints in chosen language. As in other cases, for graphical notation we add corresponding tags to reflexive association relationships (Figure 7): "Irreflexive", "Asymmetric", "Acyclic", "Intransitive", "Symmetric", and "Antisymmetric". Before defining OCL expression for acyclic constraint on reflexive relationship (the general case is represented in Figure 8) we have to introduce the transitive closure operation 0:

6. Integrity constraints on sets of objects

Equality constraint is used for groups of optional relationships or attributes. If used between two or more relationships it specifies equal dependency between them. This means that if class instance participates in one relation, so it must participate in all other relationships constrained by this constraint; instances participating only in one relation could not exist 0, 0, 0, 0, 0. Analogously, equality constraint between attributes indicates that if one of attributes has value other attributes constrained by this constraint also must do so. UML specifies equality constraint as textual constraint in a note for class. Figure 9 presents suggested extension of UML graphical notation for equality constraints. Tagged value {equn} is proposed to mark optional attributes or relationships constrained by equality constraint. Equality constraint on group of optional attributes `c1,…, cn` of object type `A` can be defined by OCL expression:

```
context A
```

```
inv: (self.c1.isUndefined() implies
 self.c2.isUndefined() … and
   self.ci.isUndefined() … and
   self.cn.isUndefined()))
 and
 (self.c2.isUndefined() implies
   self.c1.isUndefined() and
   Self.c3.isUndefined() … and
   self.ci.isUndefined() … and
   self.cn.isUndefined())
 … and
 (self.ci.isUndefined() implies
   self.c1.isUndefined() and
   Self.ci-1.isUndefined() … and
   self.ci+1.isUndefined() … and
   self.cn.isUndefined())
… and
 (self.cn.isUndefined() implies
   self.c1.isUndefined() and
   self.ci.isUndefined() … and
   self.cn-1.isUndefined())
or
```

```
(not(self.c1.isUndefined()) implies
  not(self.c2.isUndefined()) … and
  not(self.ci.isUndefined()) … and
  not(self.cn.isUndefined()))
 and
 (not(self.c2.isUndefined()) implies
   not(self.c1.isUndefined()) and
   not(self.c3.isUndefined()) … and
   not(self.ci.isUndefined()) … and
   not(self.cn.isUndefined()))
 … and
 (not(self.ci.isUndefined()) implies
   not(self.c1.isUndefined()) and
   not(self.ci-1.isUndefined()) …and
   not(self.ci+1.isUndefined()) …and
   not(self.cn.isUndefined()))
… and
 (not(self.cn.isUndefined()) implies
  (not(self.c1.isUndefined()) and
   not(self.ci.isUndefined()) …and
   not(self.cn-1.isUndefined()))
```

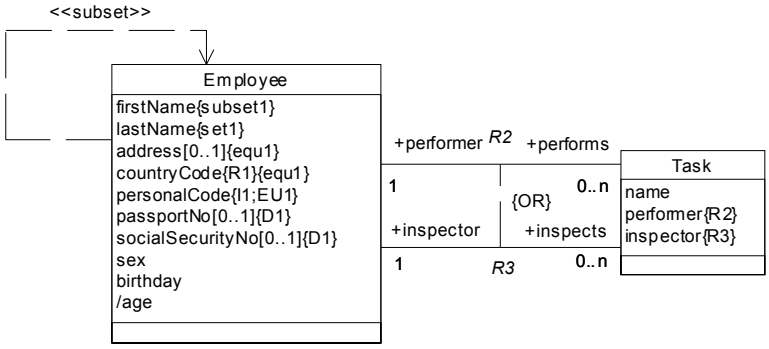

Figure 9. Example of notation of integrity constraints on set of objects

Subset constraint between two relationships indicates that set of class instances participating in both relationships is subset of class instances participating in one relationship 0, 0, 0, 0, 0. In other words, an instance of class must participate in one relationship before it can participate in another relationship. Analogously subset constraint can be defined on optional attributes. UML allows to specify subset constraints between associations by attaching the tagged value "{subset}" to the dependency arrow between the associations 0, 0 (for unification of representation, we rename "constraint label" used in original source with value of tag, defined by stereotype of constrained element). Direction of the arrow is from subset to set. However UML does not provide a graphic notation for subset constraints defined by attributes. To present these subset constraints graphically we introduce reflexive dependency for class with tag {subset} and tagged values {setn}/{subsetn} for attributes (groups of attributes) defining subset constraint on set of instances (Fig. 9). Let have object `A` with subset constraint on attributes `v1,…,vn` marked with tag {subsetn} and `p1,…, p2` marked with tag {`setn`}. This means that the set of object A instances with defined values for attributes `v1,…,v2` `is` a subset of the set of object type `A` instances with defined values for attributes `p1,…,p2`. OCL expression for this constraint can be the following:

```
context A
inv: not(self.v1.isUndefined()) and
 not(self.v2.isUndefined()) … and
 not(self.vi.isUndefined()) … and
 not(self.vn.isUndefined())
implies
 not(self.p1.isUndefined()) and
 not(self.p2.isUndefined()) … and
 not(self.pi.isUndefined()) … and
 not(self.pn.isUndefined())
```

Subset constraint on optional relationships can be expressed graphically therefore we don't need OCL expressions.

Exclusion constraint between relationships with other object types indicates that at any given time every instance of class may participate in at most one of these relationships. Analogously exclusion constraint between class attributes indicates that at most one attribute can have value 0, 0, 0, 0, 0. To indicate

this, UML uses an "or" constraint between the associations, attaching the constraint tag "{or}" to a dotted line connecting the corresponding associations. UML or-constraints can only be used between associations (Figure 9), and cannot be used between recursive associations, attributes, or between attributes and associations. For example, constraint that the same employee can be performer or inspector can be expressed as shown in Figure 9. For representation of exclusion constraint on attributes in diagrams we need to add constraint tag {orn}. The index indicates a group of optional attributes that may have at most one mandatory attribute. In general, exclusion constraint on a group of object A optional attributes c1,..,cn can be expressed using the following OCL expression:

```
context A
inv:(not(self.c1.isUndefined())implies
  self.c2.isUndefined() … and
  self.ci.isUndefined() … and
  self.cn.isUndefined())
and
(not(self.c2.isUndefined()) implies
  self.c1.isUndefined() … and
  self.ci.isUndefined() … and
  self.cn.isUndefined())
… and
not(self.ci data.isUndefined()) implies
  self.c1.isUndefined() … and
  self.ci-1.isUndefined() … and
  self.ci+1.isUndefined() … and
  self.cn.isUndefined())
… and
(not(self.cn data.isUndefined() implies
  self.c1.isUndefined() … and
  self.ci.isUndefined() … and
  self.cn-1.isUndefined())
```

Integrity constraints on sets of instances of different object types include constraints on paths on relationships and, in particular, loops 0, 0. They are the most complicated constraints that may be expressed graphically (these types of constraints are considered in Section 7). Certainly, there are constraints defined by domain expert that cannot be expressed by conventional stereotypes. The conceptual language is needed for description of these constraints, for example, OCL.

7. Integrity constraints on paths of relationships

These constraints are the most complicated constraints and are not considered at all in the most popular conceptual modelling methods. Only authors of 0 and 0 have analyzed paths of relationships trying to find out redundant, unconstrained and constrained association loops. In this section we will investigate constraints on paths of relationships showing the importance of these constraints enabling to capture important domain semantics making model more precise.

During modelling of many classes and relationships we can get relationships path from a class, through other relationships and classes, back to the same class where we have started. Such a path of relationships is called loop 0 (Fig. 10-11). It is important to find out such loops and to investigate if association loop means redundant information. Sometimes it doesn't and therefore we don't need any additional constraints, but sometimes it does and it means that these loops have specific domain rules and policies such that sets of associations are interrelated. In this situation one of association from loop requires constraint like subset or equal set constraint on path of relationships. It means that not all instances of object type can participate in relationship but just instances participating in set of constrained relationships 0.

UML doesn't have graphical representation of this constraint. In 0 and 0 the similar suggestions to display this constraint are given. The notation of constraints on relationships paths is shown in Figures 10-12.

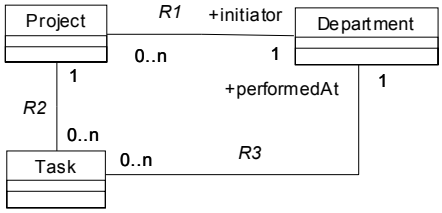

Figure 10. Not-redundant loop (the tasks of the project may be performed at different departments)

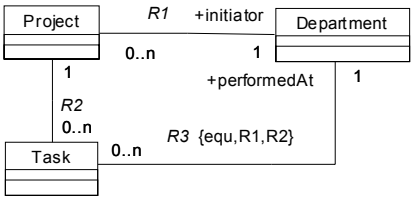

Figure 11. Redundant loop with restricted association R3

In Figure 10, the loop is not redundant as sets Department.Task and Department.Project.Task are different: tasks, belonging to Projects initiated by concrete Department, may be performed at different Department. In Figure 11, in contrast, tasks, belonging to Projects initiated by concrete Department must be performed at the same Department. In this case, the equal set constraint means that the set of instances selected by traversing a loop in one direction (Department.Task, or R3) has to be the same as the set of instances selected by traversing the loop in the opposite direction (Department.Project.Task, or R1, R2), according to the rules and policies of the domain. This constraint may be specified in OCL:

```
context Department
inv: self.Project.Task = self.Task
```

The same constraint may be expressed in a more practical way, by the principle of using for context

class having the maximum number of instances (this means that if Project is initiated by some Department, then instances of Department.Project.Task, having association path R2, R1 with instance of Department, must have association R3 with the same instance of Department):

```
context Task inv:
self.Project.initiator=self.performedAt
```

The subset constraint on path of relationships may occur in a situation, when the set of instances found in one direction is a subset of instances obtained by traversing in other direction.

The second situation is displayed in Figure 12, when we have a class with reflexive relationship and mandatory relationship with other class. In this case we have to check if reflexive relationship can associate any instances. If not, besides earlier mentioned constraints for reflexive relationship we need to add a constraint on relationship path.

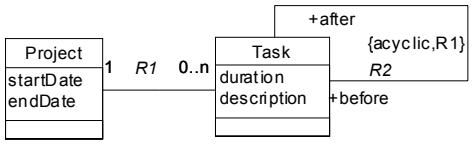

Figure 12. Example of notation for constraints on relationships path for Tasks having reflexive relationship and mandatory relationship with Project

In Figure 12 acyclic reflexive association constraint, restricted by association R1 denotes constraint on association between tasks: the concrete task cannot occur repeatedly after itself or somewhere in the sequence of tasks of particular Project (R2 {acyclic, R1}).

OCL constraint for general case of reflexive relationship for object type A having mandatory relationship with object type B (Fig. 13):

```
context A
inv: self.ra → forAll(e:A|e.B=self.B)
 and self.raClosure()→
  not(includes(self))
```

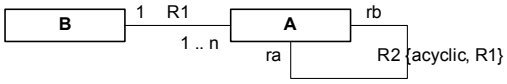

Figure 13. Example of constraints on reflexive relationship for class A having mandatory relationship with class B

8. The summary of stereotypes and tags for representation of integrity constraints

All proposed stereotypes and tags are listed in Table 1 (standard UML stereotypes as <<enumeration>>, <<dataType>> are not included here). They may be considered as potential UML profile for precise conceptual modelling including integrity constraints. Stereotypes proposed for conceptual modelling in 0, 0 have strong semantic issues for discovering true types of objects in conceptual model, but they should be merged with stereotypes for integrity constraints for obtaining precise conceptual models suitable for transformation to executable software (every kinds of schemas and code).

The variable values of tags are different for groups of instances of stereotype (as stereotypes are attached to constrained elements of group on which integrity constraint is applied). They may be derived using OCL constraints included as part of profile. We accept, for example, that <<Integrity Constraint Stereotype>> is sub typed to <<Stereotype on Integrity Constraint>> and <<Stereotype on Constrained Element>>, and names of these stereotypes are the same for the same type of Integrity Constraint:

```
context IntegrityConstrait
inv: self.constrainedElement→
 forAll(ce:Element|ce.Stereotype.name=
  self.Stereotype.name
```

For tags, meta attribute is introduced for StereotypeOnConstrainedElement:

```
context StereotypeOnConstrainedElement
def: icname:String
```

The value of tag may be derived from OCL constraint on metamodel:

```
Context Class
Let i=self.IntegrityConstraint→
 select(i:IntegrityConstraint
  |i.stereotypeName='EqualSet')→
   asSequence() in
i→forAll(ic|ic.ConstrainedElement→
 forAll(ce:ConstrainedElement|
  ce.Stereotype.name='EqualSet'
   and ce.Stereotype.icName=
    'equ'.concat(i.indexOf(ic))))
```

9. Conclusion and future work

Precise definition of conceptual model is one step on the way towards automation of transformation of business information to the physical level where its' processing may take place. In this paper UML is analyzed as the most suitable notation for constructive conceptual modelling of problem domain and software. It is well supported by many CASE tools and widely accepted as standard modelling language.

For obtaining precise conceptual model, the problematic elements are integrity constraints. Currently, they are not comprehensively studied in UML related methodology of conceptual modelling. In current practice, constraints are usually deferred to the phase of detail design. In this work, the stereotypes are proposed for extension of UML for conceptual modelling of required variety of integrity constraints selected from outstanding methods of conceptual modelling.

Table 1. Stereotypes and tags for conceptual modelling

Stereotype	Base class	Tags	Type of tagged value	Description
Association	Association	Rn	Variable string (concatenation of "R" and association number in model)	Association number (tag introduced for reference)
Constrained path of associations	Association	{R1,…, Rn}	Variable sequence (inclusion of tags of associations)	Association constrained by path of associations
Derivation	Attribute/ association	/	Constant symbol	Derived element. It must be supplemented with expression for derivation
Disjunctive mandatory constraint	Attribute/ association	{Dn}	Variable string (concatenation of "D" and number of disjunctive mandatory constraint of class/model)	Participation in disjunctive mandatory constraint on attribute/association
Equal set constraint	Attributes	{equn}	Variable string (concatenation of "equ" and number of equal set constraint of class)	Participation in equal set constraint
Equal set constraint	Association	{equ, Ri,…Rk}	Variable tuple. The first element is string "equ", and the rest elements are tags of associations of constrained path	Association derived according to equal set constraint on path of associations
Exclusion constraint	Group of attributes/ associations	{xorn}	Variable string (concatenation of "xor" and number of constrained group of attributes of class/associations of model)	Participation in exclusion constraint
External uniqueness constraint	Attribute	{EUn}	Variable string (concatenation of "EU" and number of external uniqueness constraint of model)	Participation in external uniqueness constraint on attribute. Tag is displayed beside attributes of object types constrained by external uniqueness constraint. It must be supplemented with expression for constraint with expression for constraint
Identifier	Attribute	{In}	Variable string (concatenation of "I" and number of identifier of class)	Part of identifier of class
Integrity constraint	Constraint	N/A	N/A	Integrity constraint, constraining one or more elements of UML model
Internal uniqueness constraint	Attribute	{Un}	Variable string (concatenation of "U" and number of internal uniqueness constraint of object type)	Participation in internal uniqueness constraint on attribute
Irreflexive, Acyclic, Asymmetric, Intransitive, Symmetric, Antisym-metric	Reflexive association	{irreflexive} {acyclic} {asymmetric} {intransitive} {symmetric} {antisym-metric}	Constant string	Constrained reflexive association
Primary identifier	Attribute	{P}	Constant string	Part of primary identifier of class (if it is omitted, by default the artificial primary identifier is accepted)
Referential attribute	Attribute	{Rn}	Variable string (tag of corresponding association)	Referential attribute may be not displayed in the list of attributes of class if referential attribute has no other constraints. In such case it is accepted by default as reference to primary identifier of corresponding association member (as in 0).
Subset constraint	Group of attributes/ associations	{setn/ subsetn}	Variable string (concatenation of "set"/"subset" and number of constrained group of attributes of class/associations of model)	Participation in subset constraint

In UML 2.0 version, the capabilities for extension – profiles, stereotypes, tagged values and constraints – were improved and clarified. Simple stereotypes are not adequate for representation of all types of integrity constraints; in such cases the more expressive and compact tagged values were proposed that not only serve for visualisation but also may be used for generation of database schemas and software code. Values of tags are typed; in complicated cases they are derived from elements of UML model.

The proposed list of stereotypes, tags and patterns for OCL constraints may be considered as potential UML profile for precise conceptual modelling including integrity constraints.

References

[1] BSBR: Business Semantics of Business Rules. *OMG document bei*/2004-01-04, 2004.

[2] **J. Debenham.** An analysis of Database Rules*. International Database Engineering and Applications Symposium (IDEAS '97)*, *August* 24–27, *Montreal, Canada*, 1997, 113–120.

[3] Dresden OCL toolkit, 2005*, Available at: http://dresden-ocl.sourceforge.net/index.html.*

[4] **M. Gogolla, M. Richters.** Expressing UML class diagrams properties with OCL. *Clark, A., Warmer, J. (eds.): Object Modeling with the OCL, The Rationale behind the Object Constraint Language, Springer-Verlag, London, LNCS* 2263, 2002, 85–114.

[5] **G. Guizzardi, G. Wagner, N. Guarino, M. Sinderen.** An Ontologically Well-Founded Profile for UML Conceptual Models. *In: A.Person and J.Stirna (Eds.): CAISE* 2004, *LNCS* 3084, 2004, 112-126.

[6] **G. Guizzardi, G. Wagner, M.A. Sinderen.** A Formal Theory of Conceptual Modeling Universals. *In: WSPI* '04, *Available at: http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-112/Guizzardi.pdf*, 2004.

[7] **J.L. Hainaut.** DB-Main Reference Manual. *Version* 6.5. *Dept. of Computer Science, University of Namur, Belgium*, 2002.

[8] **T.A. Halpin, A. Bloesch.** Data modeling in UML and ORM: a comparison. *In Journal of Database Management, Vol.*10, *No.*4, 1999, 4–13.

[9] **T.A. Halpin.** Integrating fact-oriented modeling with object-oriented modeling. *In: Information Modeling in the New Millennium. Idea Group*, 2001, 150–166.

[10] **T.A. Halpin.** Join Constraints. *In: Proc. EMMSAD*'02: 7*th Int. IFIP WG*8.1 *Workshop on Evaluation of Modeling Methods in Systems Analysis and Design. Toronto*, 2002, 121–131.

[11] **T.A. Halpin.** UML Data Models from an ORM Perspective (part 1–10). *In: Journal of Conceptual Modeling, No.*1, *April* 1998 *till* 10 *August* 1999, *Available at: http://www.inconcept.com*.

[12] **T.A. Halpin.** Verbalizing Business Rules (part 1-11). *In: Business Rules Journal, Volumes from* 4, *No.*4, *April* 2003 *till* 6, *No.*6 *July* 2005, 2003-2005.

[13] ISO/TR 9007. Concepts and Terminology for the Conceptual Schema and Information Base. *ANSI, New York*, 1987, 120.

[14] **I. Jacobson et al.** The unified software development process. *Addison-Wesley Professional, Boston*, 1999.

[15] **S.J. Mellor, M.J. Balcer.** Executable UML. *A foundation for model-driven architecture. Addison-Wesley, Boston*, 2002.

[16] **E. Miliauskaite, L. Nemuraite.** Taxonomy Of Integrity Constraints In Conceptual Models. *In: Proceedings of IADIS Database Systems* 2005, *http://www.iadis.org/multi2005/program_multi2005.htm*.

[17] **L. Nemuraite, B. Paradauskas.** From Use Cases to Well Structured Conceptual Schemas. O. *Vasilecas, et al. (eds): Information Systems Development: Advances in Theory, Practice and Education, Springer*, 2005, 303-314.

[18] **B. Paradauskas, I. Nemuraitė L.** Duomenų bazės ir semantiniai modeliai. *Technologija*, ISBN 9955-09-436-2, 2002, 13-45.

[19] **T. Pender.** UML Bible. *Willey Publishing, Inc, Indianapolis, Indiana*, 2003.

[20] **L. Starr.** Executable UML. How to build class models. *Prentice Hall, Upper Saddle River*, 2002.

[21] Tag Value Language. *In: UML™ Profile for Schedulability, Performance, and Time Specification, OMG document formal*/05-01-02, 2005, A1-A5.

[22] **B. Thalheim.** Entity-Relationship Modeling Foundations of Database Technology. *Springer, Berlin*, 2000.

[23] **J. Ullman, J. Widom.** A first course in database systems. 2*nd ed. Prentice-Hall*, 2002.

[24] Unified Modeling Language. *Superstructure Specification Version* 2.0. *OMG document ptc*/04-10-02, 2004.

[25] Unified Modeling Language. *OCL Version* 2.0. *OMG document ptc*/03-08-08, 2003.

[26] Using Rose Data Modeler. *VERSION*: 2001A.04.00. *The Rational Development Company*, 2001.

[27] **J.B. Warmer, A.G. Kleppe**. Object Constraint Language, The: Getting Your Models Ready for MDA. *Addison Wesley, Boston*, 2003.

DOI: 10.5755/j01.itc.34.4.12029