Customizing UML with Stereotypes

Mirosław Staroń

Blekinge Institute of Technology Dissertation Series No 2003:06
ISSN 1650-2140
ISBN 91-7295-028-5

Customizing UML with Stereotypes

Mirosław Staroń

Department of Software Engineering and Computer Science
Blekinge Institute of Technology
Sweden

BLEKINGE INSTITUTE OF TECHNOLOGY

Blekinge Institute of Technology, situated on the southeast coast of Sweden, started in 1989 and in 1999 gained the right to run Ph.D programmes in technology.

Research programmes have been started in the following areas:

- Applied signal processing
- Computer science
- Computer systems technology
- Design and digital media
- Human work science with a special focus on IT
- IT and gender research
- Mechanical engineering
- Software engineering
- Spatial planning
- Telecommunication systems

Research studies are carried out in all faculties and about a third of the annual budget is dedicated to research.

Blekinge Institute of Technology
S-371 79 Karlskrona, Sweden
http://www.bth.se

Jacket illustration:
© 2003 GillWorth gallery, www.gillworthreptiles.co.uk

Publisher: Blekinge Institute of Technology
Printed by Kaserntryckeriet, Karlskrona, Sweden 2003
ISBN 91-7295-028-5

Abstract

The Unified Modeling Language (UML) is a visual modeling language for documenting and specifying software. It is gaining popularity as a language for a variety of purposes. It was designed as a result of a unifying activity in the last decade. Since this general purpose language cannot suit all possible needs, it has built-in mechanisms for providing extensibility for specific purposes. One such mechanism is the notion of stereotype, which is a means of branding the existing model element with a new semantics. Such extended elements can then act as new model elements as if they were standard model elements. This notion is only one of the possible ways of customizations of the language. The other, more powerful technique is metamodeling, which enables to change UML by directly changing its specification.

The thesis investigates the notion of stereotype in UML both from theoretical and practical perspectives. It examines the notion of stereotype as it originally appeared in object-oriented software development as a means of branding objects according to their secondary classification in the system. The initial intent behind stereotypes is compared with the view of stereotypes in UML and similar languages, which later on provides a basis for an understanding of a stereotype in the thesis.

The thesis elaborates on a classification of stereotypes from the perspective of their usage. The classification categorizes different usages of stereotypes in different situations. Based on the classification, one such usage is evaluated in an empirical way. The evaluation is done in the form of an experiment on how the stereotypes influence the understanding of UML models. An example of a customization of UML for a conceptual database model is presented. It is a basis for a study on the expressiveness of stereotypes in the context of persistency modeling in object-oriented software. Two ways of the introduction of the stereotypes into the software development process (dependent and independent of UML tools) are outlined.

The thesis contains also a presentation of how the knowledge expressed as ontology can be imported into domain models expressed in UML. This research can be seen as a further study on the customization of UML towards usage of ontology-based knowledge.

Acknowledgments

First, I would like to express my gratitude to my advisor, *Dr. Ludwik Kuzniarz*, who provided me with the opportunity, objectives and motivation for my studies and research. I would like to show my gratitude to my secondary advisor and examiner, *Prof. Claes Wohlin*, who has devoted a significant amount of time and effort to provide me with ideas, criticism and inspiration. I would like to thank my colleagues (from, but not only, the SERL group) for their contribution with opinions, comments and help in several different ways.

I would like to thank my wife *Sylwia* for her understanding and constant support for my pursuit of knowledge during both bitter and joyful moments.

Finally, I would like to thank all the persons that are helping me with my research in various ways.

Contents

Introduction

1. Background

Software engineering is based on modeling [1]. Models can be of various kinds, each serving its own purpose. The term *model* is derived from the Latin word modulus, which means measure, rule, pattern or example to be followed. In software engineering, models are used for purposes varying from project planning, through specifications to designs and metrics. The strength of models is in their ability to present abstractions of a set of objects or phenomena. However, the creation and usage of models depends to a large extent on the consciousness of the modeler. Ludewig [1] distinguishes basically between two kinds of models – descriptive and prescriptive models. Descriptive models are used for mirroring the original phenomena, while prescriptive models are used as a specification of a thing to be created. Prescriptive models are particularly important for software engineering, since most models in software engineering are prescriptive models [1], like process models, information flow, design models, etc. For example design models are used to guide people involved towards a successful creation of a program. In software engineering, prescriptive models are built with the usage of certain languages. One of the modern, visual modeling languages is the Unified Modeling Language – UML [2]. It is a graphical language for visualizing, documenting and specifying software. The universality of UML can be seen both as an advantage and disadvantage. An advantage is that it can be used for a variety of purposes, but the disadvantage is that it is not suited ideally for each purpose. Therefore, there is a need for a customization – tailoring – of the language for specific needs. This issue is pursued in the thesis in both theoretical and practical dimensions.

1.1. Outline of the introduction

The argumentation of the thesis starts with an introduction into the area of language customization in section 2, providing the necessary definition of the notion of customization, which is a basis for further discussion in the thesis. The language customization is placed in the context of the Unified Modeling Language and the extension mechanisms in UML are outlined. One of the extension mechanisms is chosen to be elaborated on in section 3 which investigates the notion of stereotype. The section starts from the original concept of stereotypes as they were defined independently of software development methods, through the initial concept of a stereotype in software development and concluding with the investigation on how the stereotypes are perceived in the Unified Modeling Language. Section 3.1 provides an overview of the existing classifications of stereotypes to provide a context on how the stereotypes are perceived in software development. It is followed by the presentation of the notion of profile in section 3.2 as a means of grouping language extensions, which wraps up the outline of the notion of stereotype. Section 4 outlines an alternative approach to customization of the language, by providing language libraries, in this particular case by extracting the knowledge from existing formalisms into UML-encoded models.

Section 5 presents the research questions with references to specific publications. The introduction ends with an outline of further works in section 6 and the summary of the publications building up the thesis in section 7. The publications are enclosed after section 8. They are referenced to in the course of the introduction sections using Roman numbers which can be found in section 8, whereas other papers are referenced to with Arabic numbers which can be found in the reference list. In the publications by the author of the thesis, the alphabetical order of names of authors was maintained[1]. The papers included in the thesis are provided with abstracts in section 8.

2. Language customization

To provide a foundation for further discussions, a crucial concept of customization must be defined as it is subsequently used in the thesis and all the related research. Such a definition explains how the customization is understood in the course of the thesis.

> *Customization of the language is making the language easier to use in a certain context.*

The context in the definition is the purpose for which the customization is done. For example it can be a specific domain or a specific software development process. If it is the process, the customization may involve making the semantics of the language more precise (i.e. providing additional consistency rules) or providing new modeling elements specific to the process. "Making the language easier to use" means that the customization provides new modeling elements or model libraries (as indicated in [3]) containing reusable solutions for a specific context. The Unified Modeling Language is a language which enables its customization in several ways which fulfill the above definition.

The idea of one, unified language, which would be a basis for a whole family of languages, is a current issue in software engineering. Such an approach is deeply explored within the context of metamodeling and Meta Object Facility [4]. The design of the Unified Modeling Language allowing for the change of its specification by altering its metamodel provides a basis for language customization [5-7]. The perspective on UML which puts it into the role of a family of languages rather than a single unified and fixed[2] language is a starting point for extensibility of the language. The language was designed in such a way that it would enable adding new, "virtual" model constructs by the users of the language. However, the designers of the language reused for this purpose a concept which was initially not designed for this aim. It resulted in misunderstanding of one of the mechanisms for language extension and usage not according to the designers' intentions.

[1] With two exceptions; the first one is the publication based on my master thesis [VII], in which the thesis formed a great majority of the paper; the second one is the series of two publications [VII, IX] where the authors are ordered alphabetically with the exception when the contribution was not comparable with contributions of the other authors.

[2] i.e. having a fixed set of language constructs.

The view of UML as a family of languages is an approach to using modeling languages [8] in a more efficient way. The main advantage of this approach is that it considers UML as a language which is twofold. It contains the core UML constructs, and the extended constructs. The core constructs are the same in every language in the family, and the extended constructs are specific for each language in the family. Cook [8] spots the main drawbacks of UML models to be its usability in a variety of purposes with help of a fixed set of standard language elements. Since the semantics of similar constructs in different domains (for which the models are created) is not the same, the language should incorporate such variability. Examples of such two domains are multimedia systems, which stress the variability of multimedia items (modeled usually as classes) and real-time systems, which emphasize the timing constraints. The standard UML has been found to be insufficient in areas critical to the development of domain specific software. For example, some limitations for real-time systems (modeling timing constraints and implementation architecture) have been identified by McLaughlin and Moore [9]. The issues (among others) were later addressed by the Object Management Group (OMG) and resulted in the elaboration of the UML profile for performance and schedulability [10]. This and other similar initiatives lead to the perception of UML as a base language, which should be tailored for specific purposes incorporating domain-specific concepts. It can be achieved by either using the built-in extension mechanisms of the language or engineering the language with help of metamodeling approaches. One of the built-in extension mechanisms in UML is the notion of stereotype. However, to be able to fully analyze stereotypes and their usage in UML, an overview of this notion in software development in general is required.

2.1. Extension mechanisms in UML

UML has three built-in mechanisms for extensibility. Each of the mechanisms is suited for a different purpose. One of the mechanisms is constraints, which are logical expressions restricting the usage of a constrained element. If used separately from stereotypes, they restrict the usage of the instance of a model element which they are attached to [11]. For example, if attached to a class, they restrict the usage of its instances. If the constraints are defined as part of a stereotype, they provide a means of changing semantics of the stereotyped model element. Such a use of constraints is discussed in [VI].

The second mechanism, which is mainly used for the automatic model processing, and is designed for that purpose, is the mechanism of tagged values. A tagged value is some additional information which can be processed automatically by UML tools. However, over time, this mechanism evolved into more than just additional information for model processing software. The current definition of tagged values comprises the tag definition, which is a specification of the type of tagged value, its multiplicity, etc. Complemented with it, the tagged value can be used as a virtual link between model elements in the same way as if the link was an instance of an association from the UML metamodel [VI].

Finally, the main extension mechanism in UML is the notion of stereotype, which is a means of branding the existing model elements with new semantics. The

definition of a stereotype utilizes definitions of the other extension mechanisms. A notable property of the stereotypes is that they are defined by users of the language as parts of models, and once defined they are intended to be used as other UML constructs (like class, state, etc). It is stated that their definition is outside of the language. Stereotypes are also called a lightweight extension mechanism as opposed to the heavyweight extension mechanism – metamodeling.

2.2. Metamodeling as an alternative for extension mechanisms

A completely different approach to customization of the language is the so-called first-class extensibility mechanism (heavyweight extension mechanism) – metamodeling [6, 12, 13]. It is a mechanism and a technique of language engineering. Although it can also be used for language customization, it is mainly aimed at creating, defining and changing of modeling languages and as such is not discussed in the thesis. The mechanism of metamodeling is beyond the UML specification and it can be used for changing the language to such extent, that it may not be possible to find commonalities with the original UML. Significant amount of research on the issue of metamodeling and language engineering is being performed by The precise UML group (pUML) [14].

A current initiative for formalization of the metamodeling approaches to language definitions, customizations and engineering is the Meta Object Facility – MOF [4] – specification. It defines a four-layer metamodeling architecture in the context of which the UML definition is placed. The metamodeling architecture of MOF has a significant implication on the definition of UML and its customization mechanisms. The metamodeling architecture layers can be summarized as follows:

- Layer 3 (M_3): meta-metamodel, defining the language used to define languages for modeling. The MOF itself is defined at this level, and is used to define languages at the M_2 layer.
- Layer 2 (M_2): metamodel, which defines the language used for defining user created elements. The UML specification – abstract syntax – is placed at this level[3].
- Layer 1 (M_1): model, on which all the user defined models are placed. For example, the classes on UML class diagrams are the definitions of user objects, at the same time being instances of metaclasses from language definition.
- Layer 0 (M_0): user objects, on which the instances of model elements are placed. It corresponds to the level of objects that are instances of user defined classes.

In the metamodeling approaches to language customization, the changes are introduced directly into the language specification – the metamodel. Since it is the modeling of modeling languages, a lot of issues associated with language engineering are involved [5, 15, 16]. However, they are outside of the scope of the thesis. Despite its advantages and the lack of restrictions, metamodeling has a significant threat – if it

[3] It should be stated that the UML version 1.5 is not an instance of MOF, however the future releases of the languages are supposed to be.

is used without caution, the language that is engineered may no longer be considered as a UML variant. What is more, there are practical problems with UML tool support for the approach.

3. Stereotypes in software development

The entomology of the word stereotype comes from the French "*stéréotype*", from "*stéré*" coming from the Greek "solid" and "*type*" meaning a distinctive mark or sign. The Merriam-Webster dictionary gives the following definition of a stereotype:

> "*something conforming to a fixed or general pattern;* especially**:** *a standardized mental picture that is held in common by members of a group and that represents an oversimplified opinion, prejudiced attitude, or uncritical judgment*"

The most significant part of the definition, adopted by behavioral sciences, is that the stereotype is "a mental picture…that represents an oversimplified opinion prejudiced attitude, or uncritical judgment". The main fragment which should be highlighted in this definition is that it is an oversimplified opinion.

The common understanding of stereotypes is that they are a simplistic view of a group of people that share a common set of traits. An example of the widely known stereotype is the view of a young person, who is assumed to "love rock and roll and have no respect for elders" [17]. Although neither all youngsters conform to the stereotype, nor is it their primary objective in life, they are perceived as such by a certain part of the human population. Stereotypes played a similar role in software development at the time they were introduced there.

The idea of stereotyping was first introduced into software development by Rebecca Wirfs-Brock, who used the concept of stereotype to classify objects according to their "modus operandi" [18, 19]. Wirfs-Brock's original intention behind the usage of stereotypes was similar to the aforementioned view of stereotypes in other areas i.e. as a way of oversimplifying the view of objects' role or behavior. She used a fixed set of stereotypes, useful in characterizing the special roles of objects in the system. The set consists of the following stereotypes:

- *Controller*, controlling the action of other objects,
- *Coordinator,* initiating activities and tying client objects with service provider objects,
- *Interfacer,* supporting communications between external systems and users,
- *Service Provider,* performing a specific operation on demand,
- *Information Holder,* holding values that other objects ask about, and
- *Structurer,* primarily maintaining relationships.

Wirfs-Brock's set of stereotypes was based on the state of the current literature (in 1993) and the terms used by a number of researchers for characterizing the role of objects in software designs. Moreover, the way of introducing the stereotypes by Wirfs-Brock revealed some initial signs of customization of languages that supported stereotyping because her stereotypes were also aimed at enrichment of the vocabulary for describing object roles. This statement was perhaps used by the creators of visual

modeling languages, who saw the stereotypes as a mechanism of extending the base language.

An approach which is similar to Wirfs-Brock's of using stereotypes as a secondary classification of elements was adopted by the OPEN (Object-Oriented Process Environment and Notation) modeling language – OML [20]. Its designers perceived the stereotypes as:

> *"any character string of enumeration type that is used to conceptually classify a modeling element independently from inheritance or provide some other important information about the modeling element. Stereotypes provide a facility for metaclassification"*

The initial set of stereotypes in the language was restricted (c.f. [20]), although it was divided into several groups of stereotypes (for example object, class and type stereotypes). Although the stereotypes were still perceived as a secondary classification, the language included some initial indications of the dual-nature of this notion (secondary classification and metaclassification). However, the indication was not perceived in the same way as in the language which is seen as a competitor to OML – i.e. UML.

The Unified Modeling Language also contains a definition of stereotypes, but it specifies them as one of the possible extension mechanisms of the language. In UML, the stereotypes are a way of adding a new semantics to the existing model elements. They allow branding the existing model elements with new semantics, thus enabling them to "look" and "behave" as virtual instances of new model elements [2, 21]. They are no longer seen (at least directly in the specification) as a way of additional classification of model elements, according to their "modus operandi", but rather as a way of introducing new elements into the language. This perception of stereotypes contradicts with the original perception of the concept of stereotype. It is no longer aimed at providing oversimplification of elements, but at (to some extent) complication by adding new semantics to the already existing elements. On the other hand, the usage of stereotypes is not restricted to merely this purpose. The original way of using stereotypes according to Wirfs-Brock is not "illegal" with respect to the UML specification. An example of a stereotype in UML is presented in Fig. 1. It depicts the <<Interface>> stereotype, which changes the semantics of the class. Part of the precise semantics of this stereotype is that it forbids such a class to possess any instances.

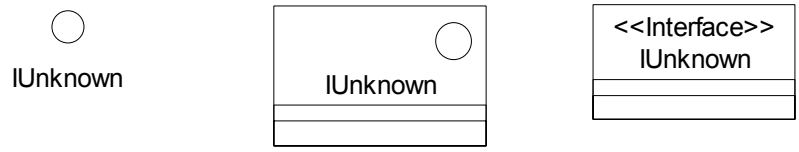

Fig. 1. Example of a UML stereotyped class that shows three ways of depicting the stereotype <<interface>>: icon, decoration of element and guillements.

During the evolution of UML (from version 1.1 [21] to 1.5 [2]), the definition of stereotypes in the UML metamodel has not changed significantly, although underwent minor revisions due to the changes in the definition of other extension mechanisms

(mostly *tagged values*, which evolved from being merely additional information for code generators in UML 1.1 specification towards virtual links between metamodel elements in UML 1.3 and later). Although at the initial stages of UML evolution the stereotypes were used in a similar way to the OML approach (as enumerations), their usage evolved into the tailoring of the language. With a growing UML tool support for this mechanism (as opposed to the lack of support for metamodeling) the stereotypes are beginning to play a major role as a means of visualizing the provision of UML as a family of languages rather than a one-fits-all modeling language.

3.1. Classifications of stereotypes

The dual nature of stereotypes in software development – their original intent of Wirfs-Brock and their definition in UML – resulted in different views and usages of this concept. To clarify the role of stereotypes in software development and to provide a systematization of the way they are used, a number of classifications were developed. The presented classifications are applicable to UML in their purpose, intent and rationale.

3.1.1. Classification of stereotypes according to their expressiveness

Berner et al. [22] examined the notion of stereotype independently from object orientation within the context of modeling languages with the focus on classification of stereotypes. Their work introduced a classification of stereotypes according to the expressiveness of the stereotype, i.e. according to the amount of changes in syntax and semantics they introduce. In their work the authors distinguished between four types of stereotypes.

1. *Decorative stereotypes*, i.e. stereotypes which do not change the semantics of a language element, but change its syntax (graphical representation),
2. *Descriptive stereotypes*, i.e. stereotypes which modify the abstract syntax of a language element and define the pragmatics of the newly introduced element without changing the semantics,
3. *Restrictive stereotypes* are descriptive stereotypes which modify the semantics of a language element,
4. *Redefining stereotypes*, which redefine a language element by changing its original semantics, w. r. t. syntax, they are similar to the restrictive stereotypes.

The classification attempts to address the complexity of a stereotype definition and provides guidelines for applying the different kinds of stereotypes. Although the authors present the examples of using the stereotypes for all types of stereotypes, they recommend a high degree of carefulness in using the redefining stereotypes. Since they allow for a complete redefinition of a model element, they should be used by experienced method specialists and forbidden for "individual engineers or within projects" [22]. Although the classification addresses the problems of how stereotypes change the extended model element, it neglects the problems of practical aspects of mechanisms for supporting stereotypes and the metamodeling levels that the stereotype definition concerns.

3.1.2. Classification of stereotypes according to their usage scenarios

A classification of stereotypes according to the metamodel level at which they are used has been introduced by Colin Atkinson et al. [23]. On one hand, the original stereotypes of Wirfs-Brock were aimed at branding objects (not classes) although they were applied to classes. On the other hand, the UML specification perceived stereotypes as a mechanism of adding new, virtual metamodel constructs, thus aimed at branding classes rather than objects. The two contradictory views of stereotypes in software development led to two alternative usages of the notion of stereotype. The classification presented by Atkinson et al. attempts to clarify the usage of stereotypes from a theoretical perspective based on the metamodel level at which they are defined and the metamodel level which they apply to. The metamodel layers addressed in the classification are the standard layers of the four-layer metamodeling architecture of MOF [4] in the context of which UML is defined. The classification identifies three stereotypes usage scenarios:

1. *Type classification*, which reflects the definition of stereotypes in UML. Stereotypes used in this way are applied to elements on the M_1-layer of the metamodeling architecture and concern the M_1-layer elements themselves.
2. *Instance classification*, which reflects the original intention of introduction of stereotypes. Stereotypes of this type are applied to the elements at the M_1-layer, but concern the instances of the branded elements at the M_0-layer.
3. *Transitive classification*, which combines the two usage scenarios. Stereotypes of this type are applied to the elements at the M_1-layer, but concern both the branded M_1-layer elements and their M_0-layer instances.

This classification suggests that the third type of stereotypes should be used with a substantial amount of carefulness and experience. This classification advises to use inheritance instead of the instance classification stereotypes, while such a usage proved itself to be useful [19]. However, it still does not categorize practical usages of stereotypes and purposes for which they are used.

3.1.3. Classification of stereotypes according to their usage

The practical usage of stereotypes in UML combines both the approaches of Wirfs-Brock and the designers of UML. Some stereotypes are defined according to the UML definition, but the commonly used stereotypes are defined, applied and used according to their original intention, or the approaches are mixed (transitive stereotypes). It is not always the case that the classifications presented above categorize, clarify and elaborate on the usage and introduction of stereotypes. Therefore, there is a need for a different classification of stereotypes, based on the usage of stereotypes in software development. Such a classification is elaborated here as a part of the research presented in this thesis. The classification is not aimed at creation of "yet another classification", but rather it is an approach to categorize the notion of stereotype within the context of practical usage and introduction of stereotypes into software development. The classification categorizes the stereotypes into three categories:

1. *Virtual Metamodel Extension Stereotypes*, which are stereotypes reflecting the notion of stereotype as an extension mechanism in UML. They are used to extend the set of UML modeling elements.

2. *Code Generation Stereotypes*, which are stereotypes aimed at making code generation rules for specific programming languages more precise and detailed.
3. *Model Simplification Stereotypes*, which are stereotypes used as a secondary classification of modeling elements.

The classification reflects the usage of stereotypes in the context of software development and UML models. The detailed description of the categories is presented in sections 3.1.3.1 – 3.1.3.3.

3.1.3.1. Virtual Metamodel Extension stereotypes

According to the UML specification stereotypes are intended to brand the existing modeling elements with a new semantics, virtually creating new modeling constructs. The semantics of stereotypes is expressed by constraints and tagged values which are defined as parts of stereotypes. The new modeling constructs defined by stereotypes should be placed in a context, which can be a technique, a process or a procedure of using them together with other stereotypes or standard modeling elements. It means that the definition of the new modeling constructs should provide a set of guidelines on how to use them. The guidelines give instructions on how to use the new elements in the same way as UML-based software development processes give instructions on how to use UML constructs to produce different models and how to make transitions from one phase (and a set of models used in the phase) to another.

Stereotypes in this category allow changing UML into another notation, and therefore, the usage of that notation – a specific method – should be defined together with such stereotypes. The majority of UML profiles fall into this category, since the category incorporates the main idea of stereotypes as an extension mechanism of UML. Examples of profiles that contain such stereotypes are: the UML Profile for Performance and Schedulability [10] or the Entity Relationship Diagram (ERD) Profile [IX]. An example of a virtual metamodel extension stereotype is the stereotype which enables to use UML classes as entities of Entity Relationship Diagrams. The stereotype – ERD Entity – is defined in the context of a database modeling process, in which the ER Diagrams are used for the conceptual database modeling phase. The definition of the ERD Entity stereotype includes the definitions of several constraints that restrict the usage of the stereotyped class with other UML model elements, for example prohibiting adding operations to these classes. Fig. 2 presents a sample class diagram which contains elements stereotyped with virtual metamodel extension stereotypes defined as part of the ERD profile – ERD Entity, ERD Entity Property and ERD Relationship.

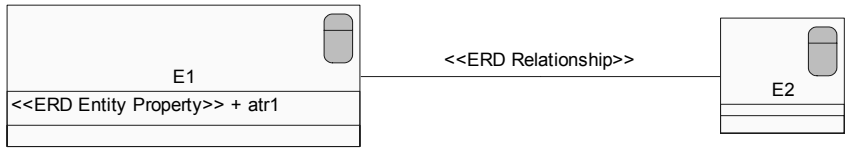

Fig. 2. A class diagram with stereotyped elements. ERD Entity stereotype is represented as an icon in the top-right corner of class' symbols.

The above diagram presents an ER Diagram, which is created from a UML class diagram by using only stereotyped elements. Such an ER Diagram should be transformed into an initial version of the logical database model, which contains elements specific for a certain database architecture (for example tables in the relational databases). The transformation rules for creation of the initial version of the logical database model from the conceptual database model are defined in database design processes. Such translation for the presented ERD Entity stereotype can be found in [II] based on [24].

3.1.3.2. Code Generation stereotypes

Certain stereotypes are intended for providing specific code generation features for the stereotyped elements. Since not all standard UML constructs can be translated directly into code in all programming languages, the precise code generation details can be specified by stereotypes, which can be interpreted by code generators. Code generation stereotypes are usually applied to modeling elements used in the late design stage, when the designs are supposed to be as closely related to the implementation as possible. Such stereotypes rarely put constraints on the usage of the stereotyped model elements in connection with other modeling elements. However, the stereotypes restrict modeling elements to have only these constructs that can be mapped into the generated code. If the standard modeling element has constructs that cannot be expressed in the generated code, then such constructs should be forbidden by constraints attached to the stereotype. On the other hand, if there is additional information that may be added to the code, but is not a part of the standard model element, then tagged values are used to add the information. An example of such a stereotype is the stereotype defined for the singleton design pattern (defined in [25]). Adding the singleton stereotype to a class results in the precise code generation for the class, as it is defined by the singleton design pattern. Fig. 3 presents a class, which is stereotyped with the singleton stereotype.

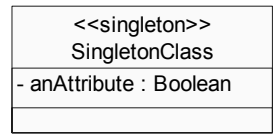

Fig. 3. A class stereotyped <<singleton>>

The standard code generation for this class in the C++ language produces the following code, which ignores the existence of the stereotype:

```
class SingletonClass
{
    Boolean anAttribute;
};
```

If the code generator is able to interpret the singleton stereotype, the code generated for the same class is:

```
class SingletonClass
{
     Boolean anAttribute;
     SingletonClass *_instance;

     SingletonClass() {}
     public:
     static SingletonClass *getInstance() {
        if (_instance == 0)
            _instance = new SingletonClass();
        return _instance;
     }
};
```

The code is complete and reflects the definition of the singleton design pattern. It is more precise than the code generated by generator unable to interpret the stereotype information.

Other examples of code generation stereotypes can be found for example in the UML Profile for CORBA (Common Object Request Broker Architecture) [26]. The profile contains elements that provide a means of precise modeling of systems which include CORBA middleware and precise generation of code for CORBA Interface Definition Language (CORBA IDL).

3.1.3.3. Model Simplification stereotypes

Not all stereotypes are intended to extend the set of UML modeling elements, nor to provide specific code generation features. Some stereotypes are intended for the simplification of models. Model simplification stereotypes are usually stereotypes used according to the original intent as a secondary classification mechanism. The semantics of such stereotypes is usually defined in a less formal way with the natural language. Moreover, since the UML specification does not impose restrictions on languages for defining stereotypes, UML itself can be used for that purpose. Since stereotypes are also used as a secondary classification of modeling elements, they can provide similar mechanisms for adding features to the stereotyped model element. Stereotypes in this category can be used as a decoration of model elements, provide the secondary classification mechanism or they can be a basis for some model transformations. Examples of such stereotypes can be the stereotypes which provide the secondary classification of objects, which are instances of the stereotyped classes. One of such stereotypes, presented in [VI] and [27] is intended to add elements – attributes, operations and an association – to the stereotyped classes. Fig. 4 presents a class stereotyped with the stereotype LDU – Logic Data Unit – from the multimedia domain [27]. The stereotype is the secondary classification of the class according to the role of the class in multimedia. The LDU stereotype designates classes which

instances are atomic units of processing in multimedia, for example audio and video units.

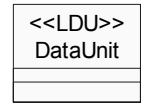

Fig. 4. An LDU stereotyped class.

The LDU stereotype is defined by elements which the LDU stereotyped class should have, which are:

- attributes:
  - data, which is an array of bytes,
  - size, which is an integer value, and
  - qos (Quality Of Service), which is a quality type (multimedia specific)
- operations:
  - read,
  - write, and
  - play
- an association to itself named granularity, which has composite and component ends.

The full definition of elements that the LDU stereotyped class should have can be depicted as a class (Fig. 5), which has the above elements.

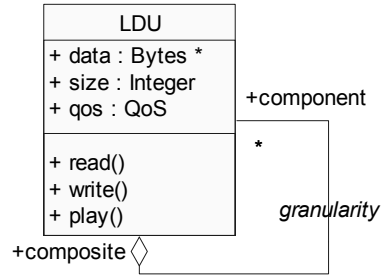

Fig. 5. The definition of the LDU stereotype depicted as a class, which has the required properties.

Although the definition of a stereotype as a class symbol is not discussed in the UML specification, it may be used, because the same properties as shown in the class symbol can be expressed as OCL constraints. The constraints can express the properties that the stereotyped model element (class in this case) should have. For instance, the OCL constraint defining the attribute "size" is as follows:

```
context LDU
inv:
  extendedElement.feature->exists(a:Attribute|a.name="size" and
  a.type.oclType()=Integer)
```

The definition of the LDU stereotype provides a means of a secondary classification of the class in the same way as inheritance. Therefore, a transformation may be defined, which adds the elements of the stereotype to the stereotyped class. After performing such transformation, the LDU stereotyped class (from Fig. 4) looks as it is presented in Fig. 6.

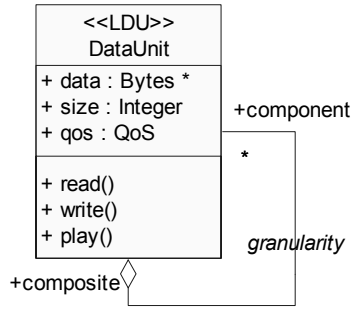

Fig. 6. The DataUnit class with the LDU stereotype specific elements added.

Defining the model simplification stereotypes does not follow the specification of stereotypes in UML directly, although such a specification can be expressed as a set of OCL constraints. Model simplification stereotypes provide a way of classifying model elements (alternative to inheritance) and as such provide means to add certain elements to the stereotyped model elements.

3.2. Profiles

Stereotypes by themselves provide a mechanism for language extension, but the real benefit from the customization of the language is evident when a certain set of stereotypes is designed to provide a way of connecting the stereotyped elements in one, to some extent complete whole. A means of packaging the stereotypes for such purpose is the notion of profile. In a strict sense a profile is a stereotyped package (<<profile>> stereotype) containing the definitions of stereotypes, constraints and tagged values [12, 13].

The notion of profile appeared for the first time in the UML specification version 1.3. Since its introduction, the profiles could provide a way of organizing UML extension mechanisms to some extent.

Firstly, they provide a means for defining new languages that are based on UML. A set of stereotypes which are mutually dependent and connected (which is expressed in a set of constraints attached to stereotypes definitions) can be closed under a specific profile. Such a profile forms a complete whole and enables using UML as some other notation. Examples of such profiles are the UML Data Modeling Profile [28] and the Entity Relationship Diagrams Profile [VI].

Secondly, the profiles allow to group elements which are not tightly interrelated, thus providing libraries of modeling elements. Such profiles do not define constraints on using the elements, i.e. there are no constraints attached to stereotypes. An example of such profile is the UML Profile for Business Modeling [2].

Nevertheless, the profiles should not be seen as libraries of models. They provide a means of building models with virtually new modeling elements, but do not contain specific solutions similar to design patterns [25].

The notion of profile has been adopted by tool vendors, since it allows for an easy extension of tool capabilities with new features. Examples of such UML tools are Objecteering [29] and Tau G2 [30]. However, the extent to which the implementation of profiles in tools is aligned with the UML specification varies from tool to tool. The description of the format for profile definition is specified in [2] and discussed in [IX].

4. Extending UML with model libraries

The idea of model libraries comes from the traditional programming languages, whose libraries play a crucial role in software development. The UML model libraries provide the same functionality as the programming language libraries or frameworks. UML tools provide dedicated libraries which contain the elements specific to a programming language or environment, for example the Java 1.2 or Microsoft .NET libraries in Rational Rose and Rational XDE products [31]. Although the construction of such model libraries is beneficial in the design phase, they are highly technology specific. Their help is evident during code generation, which is more accurately dedicated for the target environment.

Apart from the model specific libraries, there is a need and possibility to provide model libraries for a specific domain in a way that is an alternative for the technique of stereotyping and design patterns [25]. Knowledge specific to certain domains or applications has already been categorized and gathered in ontologies [32-34]. The possibility of extracting the necessary parts of it from ontologies into domain models provides a way of automatic creation of domain model libraries [IV, V, VIII, X]. The rationale behind this approach is that the library is generated by the developer, whose responsibility is to find the appropriate ontology in the ontology libraries that exists currently (for example [35]), translate it into a UML domain model and then refine it towards the specific needs of the project.

In this context, the papers concerning the extraction of knowledge from ontologies are included in the thesis to provide a more complete handling of the issue of customization of the language. Since the translation process is based on the results of the research on tool independent model processing mechanisms (introduction of stereotypes in the UML tool independent way), it provides a uniform approach to practical UML model processing techniques.

5. Research in the thesis

The research, which is presented in the thesis, provides a basis – and a platform – for introduction of customized UML into software development. It provides a theoretical as well as practical background for the customization of the language. This section summarizes the main points of the research that is presented in this thesis and presents research questions that the thesis addresses. It also presents the delimitations and outlines the scope of the thesis.

5.1. Scope

The research presented in the thesis is focused on the issue of the customization of UML and it is presented from two perspectives. The main perspective of the language customization is the usage of stereotypes. The thesis discusses the notion of stereotype in the context of UML-based software development [I]. A more general view on stereotypes, for example from behavioral sciences, is not a part of the thesis and is discussed only briefly in Section 3. The other extension mechanisms – constraints and tagged values – are discussed only in relation with stereotypes in (they are briefly described in [I], while the details can be found in [XV]). In the thesis they are perceived from the perspective of usage in defining stereotypes. The constraint language that is used throughout the thesis to specify constraints is the standard logical language recommended by OMG – the Object Constraint Language (OCL) [36]. Theoretical issues presented in the introduction and in [I] are followed by practical issues of how stereotypes can be included into UML tools. They are discussed in the thesis by presenting a UML tool independent way of introducing stereotypes into UML designs [II]. Nevertheless, possibilities of using stereotypes and the tool support for them do not show whether stereotypes can be helpful from the perspective of software developers. Therefore, an empirical evaluation of the role of stereotypes in understanding of UML models from the perspective of software developers was performed. An experiment was conducted in an academic environment to evaluate this role, which is presented in [III].

The second perspective of language customization discussed in the thesis is providing UML model libraries. Examples of such libraries are domain models constructed from existing ontologies containing knowledge about domain structure. The thesis presents research on the extraction of some parts of domain knowledge from ontologies into UML domain models. Firstly, an idea of automatic construction of domain models from ontologies is explored, corresponding constructs in ontologies and domain models are identified and a schema for an automatic knowledge acquisition process is outlined [IV]. Secondly, the acquisition process is presented in the context of a UML-based software development process, in which a place to include the knowledge acquisition process is identified. Furthermore, an improvement of a UML-based software development process by introducing the domain knowledge acquisition process is shown and illustrated with an example [V]. The benefit from the knowledge acquisition process is that even though the libraries translated from ontologies are usually specific (ontologies are usually defined for a specific purpose), they incorporate knowledge that is currently used in other fields and created by

domain experts. The knowledge usually comes from agent systems or the semantic web, where it is used to add some meaning to the structuralized data [37-39].

5.2. Research questions

The research presented in the thesis was driven by particular questions concerning certain aspects of UML customization and the usage of stereotypes for that purpose.

The fundamental question that was asked before the research in the thesis was how the customization of the language could be understood.

How can the customization of the language be defined in such a way that it addresses the context and the purpose?

An attempt to answer the question from the perspective of the research presented in the thesis was provided in Section 1.2. As indicated therein the customization of the language should be done for certain purposes and with the usage of certain mechanisms. One of the mechanisms that can be used is the notion of stereotypes, which required further investigation. It led to a narrower research question.

How to define stereotypes in a way that enables their effective usage in software development?

Answering this research question led us to the examination of the notion of stereotype presented in [VI, X, XIII]. In the research, the notion of stereotype was analyzed from the perspective of how it can be done according to the UML specification and how such definition is connected with the definition of other extension mechanisms. It was also shown how a certain set of stereotypes – ERD profile, [VI, IX] – could be defined according to the UML specification. The usage of the elaborated profile to improve a UML-based software development was also presented [I]. It was also found that the stereotypes should be classified in a way which reflects their usage. It led to another question which required further investigation.

How can the stereotypes be classified according to their usage and how does it influence the way in which they are defined?

The answer to this question was provided based on a literature study of different classifications present in the current literature and aspects of practical usage of stereotypes. It was also found that the classifications did not organize the stereotypes from the perspective of their usage in practice and the way they can be introduced into UML tools. Thus, the categorization of stereotypes according to their usage was introduced in [VI, XI] and summarized in section 3.1.3. Based on the classification, certain ways of introducing the stereotypes into software development and into UML tools were addressed within the scope of the next research question.

How can the stereotypes be introduced into UML-based software development?

The ways in which they can be introduced into usage are presented in [II, XIII]. This investigation resulted in posing some other questions, among which the most important one was:

How do stereotypes influence UML-based software development?

To provide an answer for that question an experiment was performed aimed at the evaluation of the influence of the stereotypes on the understanding of UML models presented in [III, XVI]. Although the experiment evaluated only one role of stereotypes – how the stereotypes influence understanding of UML models – the results are applicable to studies on how UML can be customized to be used as other notations (for example the Entity Relationship Diagrams). Such a study was presented in [VI, IX]. The two studies were complemented with the studies presented in [XI] which provided other examples of the remaining kinds of stereotypes.

The alternative to language customization in a way presented in the previous research question is the idea of providing a set of libraries of ready to reuse elements (partial or initial version of models). This issue was investigated as a result of another research question.

How can the knowledge from ontologies be incorporated into UML-based software development?

An attempt to respond to that question was a literature study on the possible ways of solving this problem. Although it was found that several approaches exist [37, 38, 40-43]. However, there were no solutions that enabled extracting the knowledge from ontologies into domain models without the need to extend UML for a specific purpose of ontology engineering. Since there were no solutions that could satisfy certain needs of such a method, a new way of incorporating knowledge from ontologies into domain models was elaborated [IV, V, X, VIII]. Certain requirements for that method are also presented in these papers.

5.3. Research methods

The research presented in the thesis is based on a sequence of studies. At the beginning, the identification of weaknesses was performed, which was based on investigations of existing solutions. The investigations were done via literature surveys based almost entirely on published research materials – research papers, journal articles and books. Investigations of tools capabilities were done via examinations of tools' documentations and knowledge bases available on the tool vendors' websites. They were complemented with instant feasibility studies and tests of the possibilities of tools. When the improvements were proposed, evaluations of them were done by performing tests whether the solutions fulfilled the requirements that were posed before their elaboration.

The evaluation of broader aspects of how useful stereotypes are in certain aspects was done by studying the role of stereotypes in understanding of UML models. It was done in an empirical way, by performing an experiment in an academic environment.

5.4. Contributions in the papers

The contributions in the papers consist of studies on the notion of stereotypes in UML and the analysis of the UML metamodel which defined the stereotypes [I, VI, XVI]. The studies were used as a basis for elaboration of the ERD profile placed in the context of UML-based software development and its application for improving persistency modeling in association with object-oriented software development [I]. The existing approaches to stereotyping and usage of stereotypes for such purposes as defining patterns of models [27, 44] or code generation [45] were a foundation for the elaboration of the classification of stereotypes according to their usage included in Section 3.1.3 and [VI, XVI]. Unlike other classifications, this classification was a basis for investigation of the ways of introducing stereotypes into UML designs, which categorized approaches as dependent and independent of UML tools. Such a categorization was useful in elaboration of techniques of introducing stereotypes in a UML tool independent way presented in [II]. The latter was based on the previous examination of the XMI-based model processing and verification [VII]. After analyzing and elaborating on the above issues, the research required an evaluation of some aspects of stereotyping. The aspect chosen for evaluation was the impact of stereotypes on understanding of UML models. An empirical study of this aspect an experiment was conducted in the academic environment [III]. The evaluation indicated the extent to which the stereotypes are useful and helpful in understanding of UML models.

A complement to the customization of UML with stereotypes is the customization of UML by building model libraries. It was examined by research on how the existing knowledge from ontologies could be reused in UML-based software development. The model libraries that the thesis elaborates on are UML domain models, which contain domain knowledge extracted from ontologies. Within the research in [IV, V] the notion of ontologies was investigated (languages used to define them and elements they contain). The information, which should be extracted to build correct domain models, was studied, and a solution how this information can be extracted – the knowledge acquisition process – was outlined [IV]. It was achieved by providing a means of translation from DAML+OIL (Darpa Agent Markup Language and Ontology Interchange Language) encoded ontologies into XMI encoded UML domain models (described in details in [VIII]). The knowledge acquisition process was shown to be useful in improving certain parts of UML-based software development processes [V]. Further investigations of the tool independent techniques resulted in providing a means of implementation of UML model transformations [X].

6. Future research

The current research presented in the thesis was focused on exploration of the notion of stereotypes in the context of the customization of UML. It resulted in several mechanisms and practical solutions for language customization which are ready to be used in industrial applications.

One of the possible directions of the future research is a customization of UML in such a way that it improves consistency in and among UML models. An initial investigation of sample inconsistencies in student designs within a didactic software development process can be a starting point for the elaboration of consistency rules within that process. These consistency rules can then be a basis for the customization of UML. The customization should aim at making the language constructs more precise and at providing mechanisms for automatic consistency checking within the didactic process. However, this should be preceded by further empirical investigations – supplementary case studies – on student designs to identify the most common inconsistencies and basic consistency rules for the designs.

Another possible direction is the customization of UML for a specific industrial software development process to improve certain parts of it, for example to provide a means of an automatic generation of test cases from analysis artefacts. A replication of the study presented in [III] can be performed in an industrial environment to confront the results obtained from the study in the academic setting. It may be a starting point for customization of UML for specific needs of the company, the domain and the process it involves.

Furthermore, a series of studies on how stereotypes influence other aspects of UML modeling can be performed. For example, an evaluation of the impact of using stereotypes in the design phase can be done as an empirical study both in the academic and industrial environments. Such a study can be used as a basis for a cost-benefit analysis of introducing stereotypes.

The stereotyping technique could be evaluated against its more powerful competitor – metamodeling. However, this requires a substantial amount of cases that could be studied. An experimental assessment of the two techniques can be performed, however, due to the complexity of metamodeling, a proper sample for the experiment should be found. Also the results of such a study should be treated with care, since the metamodeling technique in its current form is mainly a research playground which is only beginning to be used practically in industry.

Finally, the research on extracting domain knowledge from ontologies into UML could be extended towards such a knowledge acquisition process, which would include reasoning techniques. It would provide a more powerful solution that would extract more elements from ontologies that the current knowledge acquisition process.

7. Conclusions

The thesis examines the issues of customization of UML with the emphasis and focus on two main approaches: stereotype based UML customization and providing model libraries. The research presented in the thesis provides a platform for further research

on the particular language extensions, supplying both theoretical background and practical techniques for elaborating and introducing stereotypes into software development. Although the notion of stereotyping existed in this context for some time, it was understood in different, sometimes contradictory, ways. The research presented in the thesis attempts to categorize different aspects of usage of stereotypes and based on that elaborate ways of their effective introduction. As UML is the de facto standard in modeling languages, the research was based on it. Other languages indicated the support for stereotyping techniques, but UML was found to be the most flexible, since it supports different techniques – starting from the mere tagging of modeling elements, through the notion of stereotypes to the idea of metamodeling. The latter was not investigated in the thesis, since it is an issue slightly outside of UML, concerning modeling language engineering. The comparison of the approaches is one of the possible further directions of the research.

8. Summary of the papers

This section outlines the papers included in the thesis. The papers were referenced to by using Roman numbers. The papers included in the thesis are presented with abstracts.

8.1. Papers included in the thesis

I. Kuzniarz L. and Staron M., " Improving UML-based Software Development by Using Stereotypes for Modeling Persistency", In the Proceedings of The International Conference on Software Engineering and Applications – SEA'03, Marina del Rey, CA, 2003, pp. 301-306.

Abstract
Persistency constitutes an important issue in software development and is of particular importance within object-oriented design. However it seems that identification of persistent objects and their later design have not been given enough attention. This paper investigates one of UML-based software development processes from the perspective of modeling persistency. Drawbacks of currently used typical approaches to modeling persistent data are identified and compared with mature techniques for database modeling. An improvement of the development process by providing means for modeling of persistency at early stages and introduction of data modeling technique into UML-based development processes is proposed. The proposal is supported by a definition of a UML persistent data modeling profile based on the usage of database technology for implementation persistency. Integration of the improvement with existing solutions for realization of persistency is also discussed.

II. Kuzniarz L. and Staron M. "A Technique for Introducing Stereotypes into UML Tools", submitted to The International Conference on Enterprise Information Systems – ICEIS'04, Porto, Portugal, 2004.

Abstract
The Unified Modeling Language is a general-purpose, visual object-oriented modeling language, which can be used for a variety of purposes. However, the usage of the language for specific purposes and needs can be done by customization with the help of the built-in extension mechanisms. The customization must be supported by the tools used to produce models in the software development. This paper elaborates on the capabilities of UML tools which results in identification of some problems. The paper proposes an alternative way of introducing stereotypes, which is independent of UML tools used, based on the Extensible Metadata Interchange (XMI) format and related XML technologies. The method is compared with the introduction of stereotypes directly into UML tools by an example design.

III. Kuzniarz L., Staron M. and Wohlin C., "An Empirical Study on Using Stereotypes to Improve Understanding of UML Models", submitted to The International Conference on Software Engineering, Edinburgh, UK, 2004.

Abstract
Stereotypes were introduced into the Unified Modeling Language (UML) to provide means of customizing this visual, general purpose, object-oriented modeling language, for its usage in specific application domains. The primary purpose of stereotypes is to brand an existing model element with a specific semantics. In addition, stereotypes can also be used as notational shorthand. The paper elaborates on this role of stereotypes from the perspective of UML, clarifies the role and describes a controlled experiment aimed at evaluation of the role – in the context of model understanding. The results of the experiment support the claim that stereotypes play a significant role in comprehension of models and show the size of the improvement.

IV. Kuzniarz L. and Staron M., "Generating Domain Models from Ontologies," In the Proceedings of The International Conference on Object-Oriented Information Systems, Montpellier, 2002, Springer-Verlag, LNCS, vol. 2425, pp. 160-166.

Abstract
The paper presents and elaborates on the idea of automatic acquisition of knowledge about domain structures from ontologies into an object-oriented software development process. The information required to be included in the domain model produced during the development process is identified. The existence of the knowledge in ontologies is investigated. Requirements for ontology description languages are formulated followed by a brief evaluation of existing languages against these requirements. A schema for domain knowledge acquisition process is outlined. A realization of the schema is sketched in the paper and illustrated with an example.

V. Kuzniarz L. and Staron M., "Extracting Initial UML Domain Models from Daml+OIL Encoded Ontologies," In the Proceedings of The International Conference on Product Focused Software Process Improvement, Rovaniemi, 2002, Springer-Verlag LNCS vol. 2559, pp. 220-233.

Abstract
The paper presents and elaborates on an automatic method for creating an initial domain model using part of the knowledge contained in ontologies. It describes the method of how the initial domain model expressed in the Unified Modeling Language (UML) can be obtained in an automated way from ontologies encoded in DAML+OIL. The solution is presented in the context of the Unified Software Development Process, which uses UML as a modeling language. The elements necessary for construction of domain models are identified; a procedure for finding them in DAML+OIL encoded ontologies is described followed by suggestions for incorporation of the automatic domain model construction into a software development process.

8.2. Papers not included in the thesis

VI. Kuzniarz, L. and Staron M., "On Practical Usage of Stereotypes in UML-Based Software Development," In the Proceedings of The Forum on Design and Specification Languages, Marseille, 2002.

VII. Staron M. and Kwiatkowski J., "Verification of UML models using XMI language" In the Proceedings of The International Conference on Information Systems Modeling, Roznov, Czech Republic, 2002, pp. 190-210.

VIII. Kuzniarz L., Staron M. and Hellman E., "Generating the UML Domain Models from DAML+OIL Encoded Ontologies", In the Proceedings of the Second Conference on Software Engineering Research and Practise in Sweden, Karlskrona, 2002, pp. 78-84.

IX. Kuzniarz L. and Staron M., "Inconsistencies in Student Designs", In the proceedings of The Second Workshop on Consistency Problems in UML-based Software Development, San Francisco, CA, 2003, pp. 9-17.

X. Kuzniarz L. and Staron M., "On Model Transformations in UML-Based Software Development Process," In the Proceedings of The International Conference on Software Engineering and Applications – SEA'03, Marina del Rey, CA, 2003, pp. 391-395.

XI. Kuzniarz L. and Staron M., "Effective Usage of Stereotypes for UML Customization", presented at the Nordic Rational User Forum, Stockholm, 2003.

XII. Kuzniarz L., Staron M. and Wohlin C., "Students as Study Subject in Software Engineering Experimentation", In the Proceedings of The Third Conference on Software Engineering Research and Practise in Sweden, SERPS'03, Lund, 2003, pp. 19-24.

XIII. Kuzniarz L. and Staron M., "Stereotype Based UML Customization", Presented at The Nordic Workshop on UML-based software development, Ronneby, 2003, available at www.ipd.bth.se/consistencyUML/NordicWorkshop.

8.3. Technical reports not included in the thesis

XIV. Kuzniarz L. and Staron M., "Extracting Information about Domain Structure from DAML+OIL encoded Ontologies into UML", Blekinge Institute of Technology, Research Report 2002:02, Ronneby, 2002.

XV. Kuzniarz L. and Staron M. "Customisation of Unified Modeling Language for Logical Database Modeling," Blekinge Institute of Technology, Ronneby 2002:12, 2002.

XVI. Kuzniarz L., Staron M. and Wohlin C., "Evaluation of the Role of Stereotypes in Understanding the UML models – two academic experiments", to be published as a Research Report, Blekinge Institute of Technology.

References

[1] Ludewig J., "Models in Software Engineering - an Introduction", *Software and Systems Modeling*, vol. 2, pp. 5-14, 2003.
[2] Object Management Group, "Unified Modeling Language Specification V. 1.5", Object Management Group, 2003, www.omg.org, last accessed 2003-10-01.
[3] Smith R., "Defining the UML Kernel", *Software Development Magazine*, October 2000.
[4] Object Management Group, "Meta Object Facility (MOF) Specification V. 1.4", OMG, 2001, www.omg.org, last accessed 2003-10-08.
[5] Atkinson C., Kuhne T., and Henderson-Sellers B., "To Meta or Not to Meta - That Is the Question", *Journal of Object-Oriented Programming*, vol. 13, pp. 32-5, 2000.
[6] Atkinson C. and Kuhne T., "Rearchitecting the UML Infrastructure", *ACM Transactions on Modeling and Computer Simulation*, vol. 12, pp. 290-321, 2002.
[7] Pannaneac F., Jezequel J.-M., Malenfant J., and Sunye G., "UML Reflections", In the Proceedings of The Third International Conference, REFLECTION 2001, Kyoto, Japan, 2001, pp. 210-230.
[8] Cook S., "The UML Family: Profiles, Prefaces and Packages", In the Proceedings of The Third International Conference on The Unified Modeling Language, York, UK, 2000, pp. 255-264.
[9] McLaughlin M. and Moore A., "Real-Time Extensions to UML", *Dr. Dobb's Journal*, 1998.
[10] Object Management Group, "UML Profile for Schedulability, Performance and Time", Object Management Group, 2002, www.omg.org, last accessed 2003-09-20.
[11] Warmer J. B. and Kleppe A. G., *The Object Constraint Language : Precise Modeling with UML*. Reading, Mass, Addison Wesley Longman, 1999.
[12] Atkinson C. and Kuhne T., "Profiles in a Strict Metamodeling Framework", In the Proceedings of The Third International Conference on the Unified Modeling Language, York, UK, 2002, pp. 5-22.
[13] D'Souza D., Sane A., and Birchenough A., "First Class Extensibility for UML-Packaging of Profiles, Stereotypes, Patterns", In the Proceedings of The Second International Conference on the Unified Modeling Language, Fort Collins, CO, USA, 1999, pp. 265-77.
[14] Bruel J.-M., Clark T., Evans A., France R., Lano K., Kent S., Moreira A., and Rumpe B., "Precise UML Group", University of York, UK, 1998, http://www.cs.york.ac.uk/puml, last accessed 2003-10-08.
[15] Terrasse M.-N., Savonnet M., and Becker G., "A UML-Based Metamodeling Architecture for Database Design", In the Proceedings of Database Engineering & Applications, 2001 International Symposium on, 2001, pp. 231-236.
[16] Schleicher A. and Westfechtel B., "Beyond Stereotyping: Metamodeling Approaches for the UML", In the Proceedings of Hawaii International

Conference on System Sciences. HICSS-34, 3-6 Jan. 2001, Maui, HI, USA, 2001, pp. 10-17.
[17] Nachbar J. and Lause K., *Popular Culture. An Introductory Text*, Popular Press, an imprint of the University of Wisconsin Press, 1992.
[18] Wirfs-Brock R., "Stereotyping: A Technique for Characterizing Objects and Their Interactions", *Object Magazine*, vol. 3, pp. 50-3, 1993.
[19] Wirfs-Brock R., Wilkerson B., and Wiener L., "Responsibility-Driven Design: Adding to Your Conceptual Toolkit", *ROAD*, vol. 2, pp. 27-34, 1994.
[20] Firesmith D. G., Henderson-Sellers B., and Graham I., *The Open Modeling Language (OML) Reference Manual*. New York, Cambridge University Press/Sigs Books, 1998.
[21] Object Management Group, "Unified Modeling Language Specification Version 1.1", Object Management Group, 1997, www.omg.org, last accessed 2003-10-11.
[22] Berner S., Glinz M., and Joos S., "A Classification of Stereotypes for Object-Oriented Modeling Languages", In the Proceedings of The Second International Conference on the Unified Modeling Language, Fort Collins, CO, USA, 1999, pp. 249-64.
[23] Atkinson C., Kuhne T., and Henderson-Sellers B., "Stereotypical Encounters of the Third Kind", In the Proceedings of The Fifth International Conference on The Unified Modeling Language, Dresden, Germany, 2002, pp. 100-14.
[24] Connolly T. and Begg C., *Database Systems. A Practical Approach to Design, Implementation and Management*, 3rd ed. Harlow, Essex, UK, Addison-Wesley, 2002.
[25] Gamma E., Helm R., Johnson R., and Vlissides J., *Design Patterns : Elements of Reusable Object-Oriented Software*. Reading, Mass., Addison-Wesley, 1995.
[26] Object Management Group, "UML Profile for Corba", Object Management Group, 2002, www.omg.org, last accessed 2003-09-20.
[27] Kuzniarz L. and Piasecki M., "Modelling Basic Multimedia Notions in UML", In the Proceedings of Information Modelling and Knowledge Bases, "Frontiers in Artificial Intelligence and Applications", 2001, pp.
[28] Gornik D., "UML Data Modeling Profile", Rational Corp., Whitepaper TP162 05/02, 2002.
[29] Softeam, "Objecteering UML", Softeam, 2003, http://www.softeam.org.
[30] Telelogic, "Tau G2", 2002, http://www.telelogic.com.
[31] IBM/Rational, "Rational XDE", IBM - Rational, 2002, http://www.rational.com.
[32] Barners-Lee T., Hendler J., and Lasilla O., "The Semantic Web", *Scientific American*,
[33] Gruber T. R., "Translation Approach to Portable Ontology Specification", *Knowledge acquision*, vol. 5, 2000.
[34] Hjelm J., *Creating the Semantic Web with Rdf : Professional Developer's Guide*. New York, Wiley, 2001.
[35] "DAML Ontology Library", DARPA Agent Markup Language, http://www.daml.org/ontologies/, http://www.daml.org/ontologies/, last accessed 2003-09-26.
[36] Object Management Group, "Object Constraint Language (Ocl) Specification V. 1.4", OMG, 2002, www.omg.org, last accessed 2003-01-30.

[37] Ankolekar A., Burstein M., Hobbs J. R., Lassila O., Martin D., McDermott D., and McIlraith S. A., "DAML-S: Web Service Description for the Semantic Web", In the Proceedings of The First International Web Conference, Sardinia, Italy, 2002, pp. 348-63.
[38] Fensel D., "The Semantic Web - ISWC 2002", In the Proceedings of The First International Semantic Web Conference, Sardinia, Italy, 2002, pp. xvi+476.
[39] Ogbuji U., "The Languages of the Semantic Web", *New Architect*, June 2002.
[40] Broekstra J., Klein M., Decker S., Fensel D., van Harmelen F., and Horrocks I., "Enabling Knowledge Representation on the Web by Extending Rdf Schema", in *Computer Networks*, vol. 39, *Comput. Netw. (Netherlands)*, Elsevier, 5, pp. 609-34.
[41] Mcguinness D. L., Fikes R., Hendler J., and Stein L. A., "DAML+OIL: An Ontology Language for the Semantic Web", *IEEE Intelligent Systems*, vol. 17, pp. 72-80, 2002.
[42] Baclawski K., Kokar M. K., Kogut P. A., Hart L., Smith J., Holmes III W. S., Letkowski J., and M A., "Extending UML to Support Ontology Engineering for the Semantic Web", In the Proceedings of <<UML>> 2001, Toronto, 2001, pp. 342 ff.
[43] Cranefield S. and Purvis M., "UML as an Ontology Modelling Language", In the Proceedings of Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99), 1999, pp. N.p.
[44] Kuzniarz L. and Piasecki M., "Defining Pattern Class Stereotypes in UML", In the Proceedings of International Conference on Enterprise Information Systems, ICEIS'01, Setubal, 2001, pp.
[45] Kuzniarz L. and Ratajski J., "Code Generation Based on a Specific Stereotype", In the Proceedings of Information Systems Modeling, Roznov, Chech Republic, 2002, pp. 119-128.

Paper I

Improving UML-based Software Development by Using Stereotypes for Modeling Persistency

Improving UML-based Software Development by Using Stereotypes for Modeling Persistency

Ludwik Kuzniarz, Miroslaw Staron

Department of Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520, Soft Center
SE-372 25 Ronneby, Sweden
(Ludwik.Kuzniarz, Miroslaw.Staron)@bth.se

Persistency constitutes an important issue in software development and is of particular importance within object-oriented design. However it seems that identification of persistent objects and their later design have not been given enough attention. This paper investigates one of UML-based software development processes from the perspective of modeling persistency. Drawbacks of currently used typical approaches to modeling persistent data are identified and compared with mature techniques for database modeling. An improvement of the development process by providing means for modeling of persistency at early stages and introduction of data modeling technique into UML-based development processes is proposed. The proposal is supported by a definition of a UML persistent data modeling profile based on the usage of database technology for implementation persistency. Integration of the improvement with existing solutions for realization of persistency is also discussed.

1. Introduction

Software Development Processes (SDPs) can be perceived as a recommended sequence of engineering activities aimed at creation of a software system. In practice concrete processes are directed to a specific technology used throughout the process and also to an application domain.

The Unified Modeling Language (UML, [1]) is a well established notation enabling the production of general purpose object-oriented models. Based on the capabilities of the language *UML-based processes* were introduced. Such a process uses UML as a basic notation for expressing artifacts used and produced throughout subsequent steps of the process.

Persistency, from an object-oriented perspective, is seen as preserving an instance of the system - the state of objects constituting the system at a given point of time. The problem of modeling persistency is an important, although sometimes underestimated issue.

Database development is the process of modeling database structures (for example tables in a relational database model) and its later implementation. Database design

itself often constitutes a considerable part of the software development activity and transformation of the persistent part of the object model into databases is one of the most commonly used techniques for realization of persistency.

In the paper the object-oriented development process is examined from the perspective of how it supports modeling persistency and an improvement supporting modeling persistency is introduced. The work has been stimulated by the following observations:

- database design has well defined and proven techniques for realization of persistency which are non object-oriented,
- object-oriented development techniques provide guidelines of how to use some database techniques but only at the final stage of the development process,
- there are no means for direct addressing the problem of persistency in early stages of the object-oriented development process.

The paper investigates the current state of database modeling techniques within the context of a relational database model in UML-based object-oriented software development processes, identifies its drawbacks and proposes an improvement based on well-established database modeling techniques and UML extension mechanisms.

The paper is organized as follows. Section 2 presents results of the assessment of the current state of data modeling in UML-based software development processes, identifies its drawbacks and presents a well-established technique (process) for data modeling. It is followed by Section 3 describing UML and its mechanisms allowing customization. Then Section 4, which presents an improvement of the process, is provided. Finally a language extension is described and evaluated in the context of the existing persistency frameworks in Section 5. The paper ends with some conclusions on the improved method in Section 6.

2. Assessment of UML-based software development process

Two processes: a UML-based software development process and a relational database development process are investigated and an assessment of the first is done against the “good practices” from the second.

2.1. Process characteristics

Development processes consist of a set of phases. Each phase has goals, inputs and outputs.

The Unified Software Development Process [2] (and its variant Rational Unified Process [3, 4]) is a UML-based development process. It is both iterative and incremental. This is reflected in the way the process is designed and applied to software production. There are multiple phases in the development – Inception, Elaboration, Construction and Transition. Each phase reflects a different stage of software development. For example, the Inception phase is focused on requirements and domain modeling, but also contains a small amount of design and even testing (in

the late inception phase). Within each phase, there are several iterations over the same workflows (like analysis workflow, design workflow, etc). After completing an iteration, artefacts are incrementally updated. The result of this software development process is a working software product.

When database development is concerned, the phases of the process resemble in some way the phases the object-oriented software development processes. The first phase is the analysis, when the important concepts and their properties are identified and the relationships between concepts are found. They form a conceptual data model. The concepts are referred to as entities, since these are entities from the real world domain, which are important for the developed database. Conceptual data modeling results in a data model, expressing the concepts – entities – which should be provided with the persistency mechanisms. In later stages of the software development those entities are converted into two types of elements. One is a table (or a set of tables) which are used to store instances of this entity in the database. The other is a wrapper class, which will be used as a kind of a template for converting an instance of the entity retrieved from the database during program execution into an object encapsulating the instance.

The second phase is the conversion from the conceptual database model into a logical database model. Logical data modeling provides means to decide upon the database model used. In the relational model, data is seen as a set of tables. Each table can represent an entity and the columns in the table represent the properties of an entity. Instances of entities are stored as rows in the table. This model is focused on identifying database tables from the set of entities, transforming relationships between entities into relationships between tables. In this phase an identification of primary and foreign keys is done, types are added to properties and normalization is performed. It is important that the logical database model is not bounded to a specific DBMS (Database Management System), but the decision on database model (for example the relational database model) is already taken. Definitions of tables, relationships between them and their columns are most commonly specified during the initial data model within software development. This model is not yet bounded to any database management system, but tables have columns (which have types). Some relationships (namely many-to-many relationships) are broken into simpler elements. Data validation rules are set and the process of normalization of data is performed. So in fact the static structure of the database is created.

The next step is mapping the logical database model into the physical data model. In this phase decisions are made upon implementation environment - the DBMS - and the implementation details are elaborated (for example the decision upon Data Definition Language – DDL [5]). Additional elements, like mechanisms of stored procedures or triggers, specific to the DBMS are also added.

The last step is the implementation, and it results in the output, which is the code in Data Definition Language (for example SQL) targeted for a specific DBMS.

2.2. Assessment of persistency modeling in UML-based processes

The database development process and persistency modeling should be done as early as possible [3]. However, this principle is not realized in practice. Approaches differ

depending on the purpose – object-oriented software development or database development. From the object-oriented software development perspective, persistency is considered in the design phase – focused on database design and not on persistency modeling. The first place in the process, where the database modeling is considered, is the late elaboration phase (mid design), where tables, keys and columns come into play. What is lost is the conceptual and logical database modeling, where persistency is considered on that abstract level, regardless of the database model. The conceptual data modeling is hidden inside domain modeling, while the design phase includes physical database modeling (as specified in for example [3, 6, 7]). The problem has already been indicated in [8] and some attempts have been made to resolve it (e.g. [9]). Although the solutions exist and are used, they are still incomplete in some sense – they do not incorporate all stages of database development. As far as the UML language is concerned, the specification is independent from any database model or software development process, but involves some mechanisms for maintaining persistency (like the "persistent" tagged value). It is up to the process to use the language mechanisms for persistency. Moreover, there are persistency frameworks allowing easy transformation from a logical database model to a physical database model. The frameworks also provide flexible mechanisms for implementing the persistency mechanism and integrating it with the rest of the system.

In the current version of the Unified Software Development Process (and also other UML-based software development processes), a notion of persistency framework has been introduced. The framework is targeted at assisting developers in database development as part of object-oriented software development. One such framework, as described in [10], is focused on physical database model and run-time interoperability between the relational database and object-oriented software. It does provide a solution for modeling databases, but not particular data (and persistency). On the other hand, [5] defines a process for data modeling and database development, but it leaves several issues concerning the design of the whole object-oriented software unsolved. A step towards integration of both techniques is made in [9], but the proposed solution starts at the level of physical database modeling, without considering the conceptual or logical data models.

The frameworks used in the Unified Software Development Process come from the specification of a language profile. This profile is implemented in some UML tools (e.g. [9]), but the implementation is highly dependent on the software development process incorporated in a tool, the UML metamodel used in the tool and tool-specific extension mechanisms.

An alternative approach to the persistency framework is presented in [11]. The solution proposed there only refers to the physical database modeling. Although it introduces the notion of entity, as a database element, it actually considers entities to be something between tables and entities as considered in conceptual data models. There, an entity can contain candidate keys, which are usually considered along with tables, rather than entities [5]. Basically it presents a similar solution to [9].

An interesting consistency framework is presented in [12]. It defines an extension to a software development process, which includes database modeling. However, the database modeling presented there is targeted for Enterprise Java Beans (EJB) technology. It is well suited for that purpose, although it has rather minor significance if considered outside the EJB technology.

Based on the presented overview it can be concluded, that none of the persistency frameworks provides a solution which is based on well-established database modeling techniques incorporating widely known best practices and data modeling processes.

3. Unified Modeling Language customization

The Unified Modeling Language is a general purpose visual modeling language that is used to specify, construct, visualize and document the artefacts of the software system [13]. One of the goals behind the introduction of UML, was to make UML a tool providing a basis for a common, stable, and expressive object-oriented development method [14]. Therefore there are no special constructs that are dedicated to any particular domain of interest. Instead there is a core UML, above which the UML extensions are introduced allowing customization of the language for particular application domains (being it Real Time systems, Web development, etc.).

There are two possible mechanisms of extending the UML language, which are discussed in detail in [13, 15]. The lightweight extension mechanism, which consists of the notion of stereotype and profile (as a set of stereotypes serving the same purpose) and the heavyweight extension mechanism, which is based on altering the UML meta-model (UML syntax) based on rules of the Metadata Object Facility (MOF, [16]).

Stereotypes are a way of branding existing model elements with new semantics, thus virtually extending the set of standard UML model elements. Each stereotype is attached to a model element, which it extends – adds new meaning to it. With each stereotype, a set of constraints is associated which define the new semantics of the elements. Constraints can define changes in the way the stereotyped element interacts with other elements. For example, a constraint may prevent a stereotyped class from having attributes or operations. Each stereotype can also define a set of tagged values, which can be used as virtual properties in definition of a stereotyped model element. The detailed description of semantics and possible classifications of stereotypes can be found in [17-19]. A set of stereotypes which are defined for one purpose (like for example real-time systems or web modeling) forms a closed package – a profile. Each profile is defined by a set of stereotypes, tagged values and constraints, which constitute an integral whole. Constraints define how elements stereotyped with stereotypes from the profile interact with each other and with standard UML model elements. Furthermore, such a profile could be integrated with existing UML tools and used in practice. With each profile (or even with each stereotype) a set of model transformations can be defined. Each transformation can define how stereotyped elements could be used throughout the development process. It can show which artefacts could be produced from the stereotyped element. For example it can show how to produce an initial physical database model from the logical database model (as described in [9]).

The advantage of the lightweight extension mechanism is that it can be included into the existing UML tools. The extension mechanism is used in our approach to introduce persistency modeling extensions to the UML language and to support the improvement of the existing software development process.

4. Process improvement

The improvement of the UML-based development process is here done on two levels: process level and language level. The first is the sole introduction of early persistency modeling to the process, and the other is the notational support and tool customization for the improved process.

4.1. Early persistency modeling

To improve the software development process, conceptual data modeling should be done early during the design phase. After the phase is completed, logical and physical database models can come into play, with all the details required in corresponding phases. Such an early introduction allows taking advantage of well established methods for database development, such as usage of transformation methods (guidelines) from one model to another. Since they are well defined, all of them can be used and what is more, some of them can be automated. Another benefit of early introduction of data modeling is that a proper (and early) analysis is more likely to lead to a proper design of the database. The early introduction allows focusing on the meaning of the elements in the database, rather than the relational model or database implementation. It also provides a mean for proper separation of concerns – which elements are parts of database and which elements are parts of the object-oriented software.

Standard UML modeling elements can be used to model databases (as specified in for example [5, 9, 20]), however there are certain drawbacks while doing that.

As a basis for further discussion consider the following example. Suppose we have to develop a system, part of which is a database containing information about students, courses and enrolment of the students to the courses. Each student has to be enrolled to at least one course and each course has an arbitrary number of participants. Each student has a name and a personal number; each course has a title, a subject and a code. The conceptual data model for this fragment of the system can be expressed by the UML class diagram shown in Fig. 1.

Fig. 1. Student database model in UML

In such a diagram, the classes, which are intended to be in the database, cannot be distinguished from other classes (which are intended to be only a part of the run-time software behavior. As expressed in Fig. 1. there is no formal restrictions on these classes, so an operation can be added to either a student or a course class. This operation cannot then be transformed to the database physical model since relational databases do not store objects and their behavior.

A simple standard UML solution is to add notes to the classes, which could indicate that the classes cannot have any operations, as presented in Fig. 2.

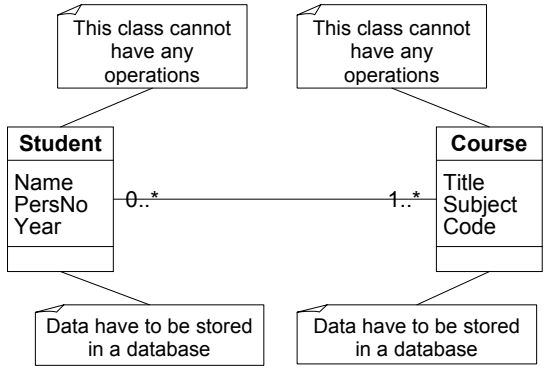

Fig. 2. Student database model in UML with notes

However, including these notes does not prevent from adding an operation to the student class. The notes only inform that this is a special class – which should be considered as persistent and stored in the database. Notes do not change the UML model element – it is still a UML class, and it will be treated by UML tools (i.e. code generator or model transformers) as a normal class. In consequence the transformation to the logical and then physical database model must be done manually. So this solution has a disadvantage that the database and other software parts are not properly separated – the database is modeled using elements which are not intended for that purpose and the transformation to physical model still must be done manually.

4.2. E-R notation in UML

The problems, mentioned in the previous section, can be avoided by using the UML language extension mechanism and providing a specific notation for conceptual and logical database modeling. The most natural candidate is the already existing and commonly used notation – Entity Relationships diagrams (ERD, [5]). The question is how to enable using this notation in UML-based software development.

The Entity Relationship (E-R) Diagram notation is used in conceptual and logical database design to model conceptual dependencies between elements from a problem domain. E-R notation is based on a number of basic concepts. Entities (also called entity types) are groups of objects with the same properties, which are identified as having an independent existence (in this sense they are similar to classes). The elements that are stored in the database are entity occurrences, which are instances of entities (like objects, which are instances of classes). Each entity has a name and a set of properties that describe the entity. E-R diagrams also contain relationships, which are meaningful associations among entities. Each relationship has two relationship ends, which are the connection points between the relationship and the entity. Each

relationship end has an optional attribute (which means that entity occurrences at this end of association can be optional) and multiplicity attribute (which indicates how many entity occurrences can take part in the relationship). The most important elements are:

- Entities,
- Relationships between entities,
- Entity properties, and
- Relationship ends.

These are not all elements existing in E-R notation, but they are chosen as the most commonly used. They are a basis for the further discussion on the notation. In the current stage of the research, the set of elements is still being enlarged.

Based on this, a database modeling profile for conceptual and logical database modeling is proposed as an extension of the standard UML language. The profile makes UML "look and behave" as the E-R notation. The appearance is achieved by using the graphical representation of stereotypes. Constraints included in the profile introduce new semantics for stereotyped classes and associations and allow using them as appropriate notions from the E-R notation. For example, there are constraints preventing the adding of operations (methods) to classes stereotyped as E-R entity. Tagged values associated with stereotyped elements provide an entry point for automatic transformations from the E-R diagram (with entities, properties, etc) to a logical database model (with tables, columns, etc).

Introduction of these stereotypes allows using a UML class diagram as if it was an E-R diagram. Taking into consideration the student database example, the resulting conceptual database model can look as in Fig. 3.

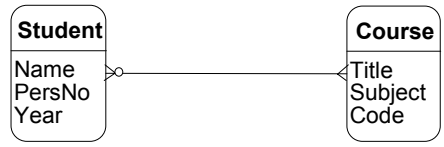

Fig. 3. Student database model using stereotyped elements

The graphical representation of E-R elements makes it clear that these concepts are intended to be used as database elements. It also distinguishes them from other elements. Constraints attached to the stereotyped elements (see the next section) prohibit activities not allowed in E-R (for example adding operations to any of the classes). The stereotype contains also information for model transformers (like the code generator), that these classes should be treated separately. In the case of the stereotypes included in this profile the information for the code generator is that there should be no code generated in an object-oriented implementation language but instead the stereotyped classes should be transformed to logical database model elements. If the target database model is relational, then a set of tables and relationships between tables should be generated.

4.3. Database modeling profile

The starting point for improving the object-oriented software development process is the customization of UML for conceptual database modeling – the ERD profile. It does not cover the whole E-R notation but it integrates with existing solutions for logical and physical database modeling. The profile described here contains two main parts:

- stereotype definitions, which extend the set of standard UML model elements (make stereotyped elements "look" like E-R diagram elements), and
- consistency rules, which enforce the E-R notation restrictions and make stereotyped UML elements "behave" as E-R diagram elements.

A full profile definition can be found in [18] and it provides also a description of how elements of this profile could be mapped to elements of physical data modeling profiles.

4.3.1. Definitions of stereotypes

To introduce the idea of the approach discussed here, the definition of the ERD_Entity stereotype is presented. The stereotype can be applied to classes. It changes the usage of the class, its graphical representation and imposes some constraints on the class to "convert" it to E-R entity. The stereotype is defined in tabular form (as recommended in [1]) in Table 1.

Stereotype	*Base class*	*Tags*	*Constraints*
ERD_Entity	Class	UNIQUE	ERD_a - ERD_h

Table 1. Excerpt from the definition of ERD profile

Constraints (ERD_a to ERD_h) are specified in the Object Constraint Language (OCL, [21]), which is a part of the UML specification. Below, example constraints (ERD_a to ERD_d) are quoted. Each constraint is preceded with a brief explanation of the intention of the constraint.

- ERD_a: An entity must not have any operations, as the entity does not have any behavior:

```
Context ERD Entity
Inv: extendedElement.operations-> Empty
```

- ERD_b: an entity can have binary assciations only with entities:

```
Context ERD Entity
Inv: extendedElement->associations->
 forAll (a| a.connection->size=2 and a.allConnections->
 forAll (r| r.type.stereotype = "ERD Entity")
```

- ERD_c: an entity can be connected only by means of relationships:

```
Context ERD Entity
Inv: extendedElement->associations->
 forAll (a|a.stereotype = "ERD Relationship")
```

- ERD_d: an entity can only contain attributes that are stereotyped as entity properties:

```
Context ERD Entity
Inv: extendedElement->attributes->
 forAll (a| a.stereotype = "ERD Entity Property")
```

There is also a graphical symbol associated with the ERD_Entity stereotype. It is defined in Table 2.

Stereotype	*Icon*
ERD_Entity	

Table 2. Icon definition table

The rest of the constraints, tags and remaining stereotypes can be found in the profile definition [18].

4.4. Remarks on technical realization

An important practical aspect of profiles is their introduction to UML tools used in software development. A complete solution on how to add the new profile to one of the popular UML tools – Rational Rose is presented in [18]. Rational Rose was chosen as an example tool to convey the experiment because the physical database modeling profile was already implemented in this tool, and the current efforts were on automatic integration of these two profiles to provide a complete solution to data modeling within object-oriented software development. An introduction of a profile into a tool requires additional effort, since tools do not fully comply with the UML standard – especially to the UML meta-model. Some tool-specific language constructs must be used to implement the constraints. In Rational Rose, the language used to implement the constraints (OCL) is RoseScript, which is a dialect of Visual Basic. Customization of Rational Rose allowed to provide a set of new modeling elements in class diagrams, a menu item entry which enables checking whether the constraints of the profile are met (if the constraints were not met, indications are given in which place profile constraints were violated).

5. Related work

The problem of mismatch between database modeling and object-oriented software development techniques has been identified in a number of publications concerning development processes [3, 9, 11, 22], tools [9] and methods [5, 8, 20]. Most of the authors suggest solutions mainly targeted at run-time interoperability between object-oriented software and relational databases (see section 2.2.2). The approach suggested in [8] also discusses issues concerning the design of a proper persistence framework.

However, it leaves the database design issues still remain unsolved. It proposes a method for defining frameworks, but not for modeling databases.

Some indications on how the UML language can be used to model databases can also be found (for example [3, 5]).

One of the earliest extensions of UML for database modeling was presented in [23]. The extension is dedicated to modeling databases with UML, with stereotypes for physical database modeling. However, the paper does not focus on the data modeling process, but rather on modeling relational databases, which makes the approach quite different from the improvement presented in this paper. In a later paper [24] the same author proposes a slightly different approach to a similar problem - soft-coded data values. The approach is based on design patterns, but still does not focus on database development.

An interesting approach to include the aspects oriented paradigm in E-R modeling is presented in [25]. The focus is on the integration between E-A (Entity-Aspect) models and E-R models. It is interesting since it discusses an introduction of conceptual database modeling into aspect oriented technology.

6. Conclusions

This paper assessed two current UML-based processes from the perspective of data modeling. Mismatches between the expected and desired data modeling processes were identified. The main mismatch is that the conceptual data modeling is not included in modern, object-oriented software development processes based on UML.

The paper also proposes to improve the software development process by including conceptual data modeling directly into the process. This is followed by an introduction of an extension of the UML language (by means of a database modeling profile) to enabling direct usage of the notion of E-R diagrams at a very early stage of development. An integration with other profiles (for physical database modeling), at the same time redirecting the reader to details is outlined. Customization of a UML tool to introduce the database profile is highlighted. Description of the theoretical basis for customization, technical details for specification of the stereotypes and the implementation code for the profile can be found in [18].

The most important advantages of the proposal are the following:

- Introduction of matured techniques from database development to the object-oriented development process,
- Integration of the existing database persistency frameworks with the object-oriented development process,
- Improvement of the perception of artefacts produced within object-oriented development processes by better separation of concerns,
- Provision of means for generation of a logical data model from a conceptual data model.

Further research is focused on integration of the proposed extensions with other UML profiles and plugging the extensions into UML tools.

References

[1] Object Management Group, "Unified Modeling Language Specification V. 1.5", Object Management Group, 2003, www.omg.org, last accessed 2003-10-01.

[2] Booch G., *Unified Software Development Process*, Addison-Wesley, 1998.

[3] Rational, "Rational Unified Process Documentation", Rational Corp, 2000, www.rational.com, last accessed 2003-01-25.

[4] Kruchten P., *The Rational Unified Process : An Introduction*, 2nd ed. Reading, MA, Addison-Wesley, 2000.

[5] Connolly T. and Begg C., *Database Systems. A Practical Approach to Design, Implementation and Management*, 3rd ed. Harlow, Essex, UK, Addison-Wesley, 2002.

[6] Arlow J. and Neustadt I., *UML and the Unified Process, Practical Object-Oriented Analysis & Design*, 1 ed, Adison-Wesley, 2002.

[7] Larman C., *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, 2nd ed. Upper Saddle River, NJ, Prentice Hall PTR, 2002.

[8] Ambler S., "The Design of Robust Peristence Layer for Relational Databases", Ambysoft, 2000, www.ambysoft.com, last accessed 2003-09-30.

[9] Rational, "Rose Data Modeling Profile", Rational, 1999, www.rational.com, last accessed 2002-10-01.

[10] Larman C., *Applying UML and Patterns*, Adison-Wesley, 2001.

[11] Ambler S., "Persistence Modeling in UML", *Software Development Magazine*, August 1999.

[12] Fowler M. and Kobryn C., "Customizing UML for Fun and Profit", *Software Development Magazine*, July 2002.

[13] Booch G., Rumbaugh J., and Jacobson I., *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

[14] Booch G., "Quality Software and Unified Modeling Language", Rational, 2000, www.rational.com, last accessed

[15] Cook S., "The UML Family: Profiles, Prefaces and Packages", In the Proceedings of UML 2002, 2000, pp. 255-265.

[16] Object Management Group, "Meta Object Facility (MOF) Specification V. 1.4", Object Management Group, 2001, www.omg.org, last accessed 2003-10-08.

[17] Gogolla M. and Henderson-Sellers B., "Analysis of UML Stereotypes in the UML Metamodel", In the Proceedings of UML 2002, Dresden, 2002, pp. 84-99.

[18] Kuzniarz L. and Staron M., "Customisation of Unified Modeling Language for Logical Database Modeling", Blekinge Institute of Technology, Ronneby, Research Report 2002:12, 2002.

[19] Atkinson C., Kuhne T., and Henderson-Sellers B., "Stereotypical Encounters of the Third Kind", In the Proceedings of UML 2002, Dresden, 2002, pp. 100-114.

[20] Gornik D., "UML Data Modeling Profile", Rational Corp., Whitepaper TP162 05/02, 2002.
[21] Object Management Group, "Object Constraint Language (Ocl) Specification V. 1.4", Object Management Group, 2002, www.omg.org, last accessed 2003-01-10.
[22] Ambler S., "Data Modeling 101", Agile Data, 2002-2003, www.agiledata.org, last accessed 2003-02-20.
[23] Blaha M. and W P., "Using UML to Design Database Applications", *Rose Architect*, April 1999.
[24] Blaha M., "Pattern for Softcoded Values", *IEEE Computer*, May 2002.
[25] Lee J.-W., Hong J.-H., and Baik D.-K., "Database Conceptual Modeling for Integrating E-a Model and E-R Model", In the Proceedings of, 2001, pp. 157-168.

Paper II

A Technique for Introducing Stereotypes into UML Tools

A Technique for Introducing Stereotypes into UML Tools

Ludwik Kuzniarz, Miroslaw Staron

Department of Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520, Soft Center
SE-372 25 Ronneby
Sweden
(Ludwik.Kuzniarz, Miroslaw.Staron)@bth.se

The Unified Modeling Language is a general-purpose, visual object-oriented modeling language, which can be used for a variety of purposes. However, the usage of the language for specific purposes and needs can be done by customization with the help of the built-in extension mechanisms. The customization must be supported by the tools used to produce models in software development. This paper elaborates on the capabilities of UML tools which results in identification of some problems related to introduction of stereotypes. The paper proposes an alternative way of introducing stereotypes, which is independent of UML tools used, based on the Extensible Metadata Interchange (XMI) format and related XML technologies. The method is compared with the introduction of stereotypes directly into UML tools by an example design.

1. Introduction

The Unified Modeling Language – UML [1] – is a general purpose, visual modeling language used for specification and documentation of software artefacts. The intention to make the language suitable for many purposes is reflected in the design of the language, which contains a fixed set of model elements and mechanisms that allow extending the language. One such extension mechanism is the notion of stereotype, which allows branding the existing model elements with a new semantics and therefore customizing the language for specific purposes. The technique of stereotyping has proved itself to be useful in several contexts. Firstly, to simplify designs by providing a secondary classification of objects according to their responsibilities in the designed system [2, 3]. Secondly, as an extension mechanism, which allows to customize the UML language for a specific purpose [4-7]. Despite its usefulness, the technique poses certain practical problems when introducing UML stereotypes into software documents specified in the UML. The problems arise from the fact that stereotypes incorporate two levels of conceptual modeling (they are specified at the metamodel level and are applied at the model level) and that support in UML tools is not sufficient. The usage of stereotypes in practice depends on the capabilities of UML tools to support them. Not all UML tools provide the same

mechanisms of introducing stereotypes and using them. Therefore, there is a need for a technique of introducing stereotypes and performing model transformations based on stereotypes in a way that is independent of UML tools. This paper presents such technique based on the Extensible Metadata Interchange format (XMI) and other XML related technologies, which overcome most of the problems that arise when using stereotypes in UML tools.

The argumentation in the paper starts with an elaboration on the capabilities of two sample UML tools to effectively support stereotypes. Subsequently, the method for introducing the stereotypes in a UML tool independent way is introduced and evaluated. Finally, an example is presented, followed by final remarks on the method.

2. UML tool support for stereotyping

With the introduction of the Unified Modeling Language the role of stereotypes was broadened (as compared to the original intention of stereotyping) and they became one of the extension mechanisms of the language. This new role posed also some problems with the practical introduction of the stereotypes into software designs. The problems were caused by the limitations of the UML tools to support the stereotypes. An analysis of the metamodel excerpt which specifies the UML extension mechanisms [5, 7-9] was used as a basis for elaboration of the requirements for the UML tools presented in [7]. An evaluation of tools against these requirements found that the existing UML tools are limited and do not provide solutions portable between the UML tools and that there is an insufficient support for stereotype based model transformations of UML models. The transformations are a way of using stereotypes to make the language more precise and efficient for certain purposes [10].

2.1. Introduction of stereotypes directly into UML tools

Including stereotypes in development processes is a special case of model transformation. Introduction of the extension of UML based on stereotypes can be done in two ways. One is the direct inclusion of stereotypes in the UML tool, which involves tool-specific extension mechanisms. The other, more universal is independent from UML tools (referred to as the tool independent technique in this paper). It is based on XMI (XML Metadata Interchange, [1]) and XSLT (Extensible Stylehseet Language for Transformations, [11]) related technologies. Naturally, it requires the support for XMI in the used UML tools.

Implementation of transformations can be done directly in the modeling tools, provided that they have the necessary features, which are:

- access to the repository containing UML models inside the tool, and
- a programming language to write scripts implementing the transformation.

An example of a UML tool is Rational Rose [12], which provides a mechanism – Rose Extensibility Interface – that allows access to models in the repository and provides a scripting language – Rose Script – to manipulate the models.

Transformations are performed directly on the UML model being edited in the tool and are accessible from a menu of the tool. Implementing such transformations requires additional skills and knowledge about the tool structure from developers. Such an approach to transformations has been presented partially in [9].

A problem with tool dependent transformations is their dedication for the particular UML metamodel (UML abstract syntax specification) used within the tool. This should have no impact on implementation, as the UML specification is only one, but individual tool vendors decided to alter the metamodel of a language and adjust it to their needs, which has been the case of the Rational Rose [12] and Telelogic Tau G2 [13] tools. What is more, as the implementation is bound to a tool, it may not, and in many cases is not, be portable across different UML tools.

2.2. Tool support

The UML specification provides a suggestion for the way of customizing the language. Theoretically, it can be extended to be alike other notations. In practice, the UML tools restrict the usage of the proposed extension mechanisms. Most of the UML modeling tools provide a certain way to extend its capabilities. The following sections provide a short overview of how stereotypes are supported in two UML tools.

2.2.1. Rational Rose

In the case of Rational Rose [12], the main mechanisms allow to:

- add stereotypes with their graphical presentation,
- specify constraints in a scripting language,
- define tagged values, and
- use XMI to import or export XMI encoded UML models

Although the tool seems to be a flexible solution for UML customization, it has some restrictions:

- introduction of new stereotypes requires restarting the tool,
- the repository does not exactly match the UML metamodel,
- lack of support for OCL constraints, although the OCL can be used, such code is not processed,
- application of no more than one stereotype for a single model element at a time is allowed,
- the set of possible tagged values types is restricted to primitive types and enumerations, which does not allow to use them according to OMG's intentions (i.e. as virtual links between elements on the metamodel level), and
- the XMI format used in the tool is modified to support diagram interchange information.

Rational Rose provides a good platform for model processing and model transformations based on stereotypes, but the usage of the tool extension mechanisms requires substantial additional knowledge on the specifics of the tool.

2.2.2. Telelogic Tau

Telelogic Tau G2 [13] is aimed for support of the upcoming UML 2.0, so its capabilities and restrictions are different than those of Rose. The main mechanisms for customization allow to:

- add stereotypes without the need to restart the tool,
- apply more than one stereotype for a single model element,
- use graphical representation of stereotypes (icons),
- use scripting language for model manipulation, and
- use unlimited (with respect to the UML metamodel) set of allowed tagged values types; tagged values are virtual links at the metamodel level.

The tool is well-suited for model processing and its metamodel is closely related to OMG's specification. However, it has some restrictions:

- lack of OCL support – the constraints can be written, but are not processed by the tool, and
- partial XMI support (import only)

The usage and introduction of stereotypes in this tool requires a different approach that the previous one, and although it is compliant with OMG's specification of stereotypes to a major extent, it requires substantial knowledge about the structure of the tool repository from developers of stereotypes.

3. Tool independent technique

Although the UML tools provide a substantial amount of possibilities to extend their capabilities, most of them are proprietary solutions (excerpt from the open source initiatives, for example the ArgoUML tool [14]). An alternative approach to the introduction of stereotypes is to use UML-tool independent technologies and integrate them with UML tools.

3.1. Tool independent introduction schema

One of the possible tool independent technologies is the XML technology. To take full advantage of it, some prerequisites should be fulfilled. The stereotype definition must be expressed as an XSLT document, so that certain parts of this document will be compatible with XMI. This document contains the semantic of the stereotype. The UML tools that are involved in the development process must support at least a basic representation of stereotype – guillements. They also must be able to export UML models to XMI documents. If the above conditions are met, then the inclusion can be done automatically in the way presented in Fig. 1:

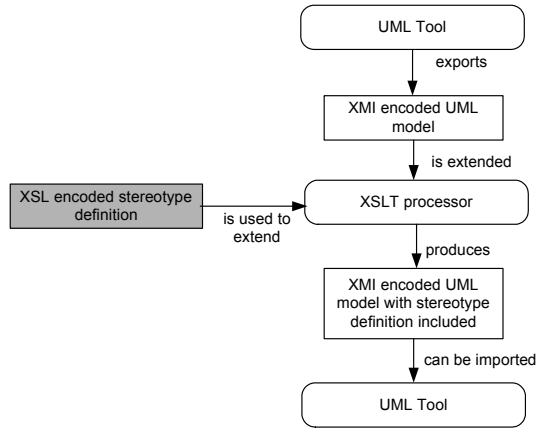

Fig. 1. Process of including UML stereotypes with XML technology

The UML tool exports a model, which contains elements stereotyped to an XMI document. The XMI encoded model is an input to the XSLT processor. The XSLT processor applies the XSLT Stylesheet with the definition of stereotype to the XMI encoded model. It produces the output model, in which the stereotyped elements contain the semantics of the stereotype. This model can be imported to the UML tool.

3.2. Checking constraints

Since the UML definition of the notion of stereotype involves the definition of constraints, their support plays a crucial role in using stereotypes. The lack of a direct support for constraint checking in the presented technique could be its major disadvantage. However, this technique can be accompanied by a constraint checking tool – xlinkit [15]. The tool allows checking constraints on UML models by the UML metamodel (i.e. well-formedness rules), and it can be used to check constraints that are a part of the stereotype definition [16]. The language used in the xlinkit tool is an XML encoded first order logic language similar to OCL. The UML models are encoded as XMI documents.

4. Example

As an example of using the method, an introduction of the sample profile for conceptual data modeling is presented. It consists of two parts, the first is a short description of the contents of the profile, its purpose and usage; the second is the usage of XML related technologies to introduce the profile into UML designs, checking the constraints and integrating the profile with the UML data modeling profile available in some UML tools.

4.1. ERD Profile

The database modeling process consists of three phases

- conceptual data modeling, which is done with usage of Entity-Relationship Diagrams (ERD),
- logical data modeling, which uses database architecture specific concepts (for example tables and relationships for the relational database architecture), and
- physical database model, which is targeted towards a specific Database Management System (for example Oracle)

The latter two steps are parts of the UML-based software development as standard data modeling profiles [17-20]. However, the conceptual database modeling requires further extensions of UML. Its introduction allows using a variant of UML class diagrams as ER diagrams and provides means of automatic generation of an initial version of the logical data model expressed as another UML class diagram stereotyped with another set of stereotypes. The ERD profile contains stereotypes which change UML classes into Entities, UML associations into Relationships, etc. The details of the profile can be found in [9]. In this paper, only one stereotype – changing a UML class into an ERD entity (<<ERD Entity>>) is presented and elaborated on in a simplified form.

The <<ERD Entity>> stereotype changes the graphical notation of the class and adds several constraints restricting the usage of the stereotyped classes. Classes stereotyped with <<ERD Entity>> should be, at the later stage, transformed into tables in the logical database model if the relational database architecture is chosen.

An example class stereotyped with this stereotype is presented in Fig. 2. The stereotype is represented in a graphical form by the icon in the right-hand corner of the class symbol (for <<ERD Entity>> stereotype) and as guillements (for <<ERD Entity Property>> stereotype).

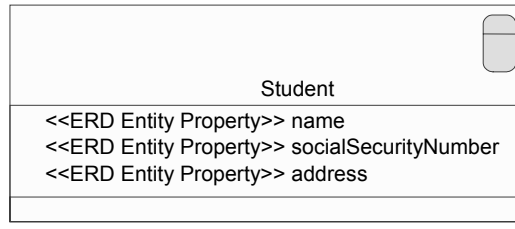

Fig. 2. An example of a class stereotyped <<ERD Entity>>

As an entity, this class is restricted according to the definition of an entity in [21]. Some of the restrictions are that the stereotyped class has no operations, it contains only a special kind of attributes – ERD Entity Properties, and the types of the properties are not specified. The first restriction can be expressed as the following OCL constraint:

```
Context ERD Entity
Inv: extendedElement.operations->Empty
```

The constraint can be checked directly in the UML tools, but it requires additional code in the tool specific scripting language. For the Rational Rose tool, the constraint can be implemented in Rose Script as:

```
Public Function NoOperations (theClass As RoseClass) As Boolean
  If theClass.Stereotype = "ERD Entity"
   Then
     NoOperations =
       (theClass.Operations.Count = 0)
  End if
End Function
```

The code is possible to be executed in Rational Rose and cannot be used in other UML tools, for which another language must be used for implementation.

To overcome this problem, the constraints may be implemented in the aforementioned xlinkit tool, which is used for checking constraints in XML documents. The tool has been tested to support checking the well-formedness rules, defined in the UML metamodel for UML models. The same constraint as previously is expressed as follows in the language accepted by xlinkit:

```
<consistencyrule id="NoOperations">
<forall var="cl" in="//UML:Class[@xmi.id]">
<implies>
  <equal op1="contains(
   '//UML:Stereotype/@extendedElement',
   '$cl/@xmi.id')" op2="0" />
  <not>
   <exists var="op" in="$cl/UML:Classifier.feature/UML:Operation"/>
  </not>
</implies>
</forall>
</consistencyrule>
```

The xlinkit implementation is more readable for a person who is familiar with the OCL syntax. It resembles an XML encoded OCL expression.

4.2. Integration with UML Data Modeling Profile

Checking of well-formedness of the stereotyped element is necessary, but the usage of the tool independent technology becomes even more evident when it is considered as a platform for transformation of models based on stereotypes. In the ERD profile example, such transformation is taking one step further in the database design process – moving to the logical data model. There are specific rules what has to be done to translate each entity, relationship, etc. to an element in the logical data model (c.f [21]). In case of entities, the transformation has to exchange the stereotype to tables and properties into columns. The Rational Rose implementation of the entity transformation rule is as follows:

```
Sub TransformClass (sClass       As Class, _
                                  ByRef dClass  As Class, _
                                  ByRef theDiag As ClassDiagram)
```

```
 ' local – temporary variables
Dim currAttr        As New Attribute
Dim associations    As New
     AssociationCollection
Dim currAssoc       As New Association
Dim newAssoc        As New Association
Dim opRole                 As New Role
Dim opClass         As New Class

   ' creation of the class representing a newly created table
Set dClass.name = "T_" & sClass.name
dClass.Stereotype = "Table"
dClass.Persistence = true

  ' copying all attributes
For i = 1 To sClass.Attributes.Count
  Set currAttr = sClass.Attributes.GetAt(i)
   Set newAttr = dClass.AddAttribute (currAttr.name, currAttr.Type,
currAttr.InitValue)
   Next i
End Sub
```

In the tool independent technique, the XSLT Stylesheet is used for performing transformations. The part of the stylesheet that is responsible for the transformation rule has three main parts. The first part is responsible for introducing the new stereotype <<Table>> into the document and for adding references to the newly created tables with it. The text in bold designates the XSLT specific code and the code in italics designates elements specific to XMI code of the stereotype definition and its references to the stereotyped elements. The standard Stylesheet elements and the template for copying elements are not included in the presented code, they can be found in [11].

```
<xsl:template
 match="UML:Namespace.ownedElement">
<UML:Stereotype
 name = 'Table' visibility = 'public'
 isSpecification = 'false'
 isRoot = 'false' isLeaf = 'false'
 isAbstract = 'false' icon = ''
 baseClass = 'Class' >
 <xsl:attribute name="xmi.id">
    <xsl:value-of select="generate-id()"/>
 </xsl:attribute>
 <xsl:for-each select="//UML:Stereotype [@name='ERD Entity']
     /UML:Stereotype.extendedElement">
  <UML:Stereotype.extendedElement>
    <Foundation.Core.ModelElement />
    <xsl:attribute name="xmi.idref"
     select="concat(
       'Foundation.Core.ModelElement
         /@xmi.idref', '__table') "/>
  </UML:Stereotype.extendedElement>
 </xsl:for-each>
</UML:Stereotype>
     <!--copy other child elements of
       UML:Namespace.ownedElement-->
</xsl:template>
```

The references to identifiers of elements in the XMI document (tag attributes xmi.idref) used in the code are created statically, based on the assumption that each newly created class stereotyped <<Table>> has an identifier which was created from the identifier of the entity that was transformed by adding the suffix "__table". The second part of the transformation stylesheet converts the entities into tables. Its implementation looks as follows (rules for bold and italics as in the previous example):

```
<xsl:template match="UML:Class">
 <xsl:variable name="id">
  <xsl:value-of select="@xmi.id" />
 </xsl:variable>
 <xsl:choose>
 <xsl:when test="count(
   //UML:Stereotype[@name='ERD Entity'
    and
   contains(@extendedElement,$id)]) > 0">
 <xsl:call-template name="ConvertToTable" />
 </xsl:when>
 <xsl:otherwise>
  <xsl:call-template name="copy" />
 </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<xsl:template name="ConvertToTable">
  <UML:Class>
   <xsl:attribute name="xmi.id">
     <xsl:value-of select="concat(
      '@xmi.id', '__table')"/>
    </xsl:attribute>
    <xsl:for-each select="
      @*[name() != 'xmi.id']">
    <xsl:call-template name="copy" />
   </xsl:for-each>
  <UML:Classifier.feature>
  <xsl:for-each select="UML:Attribute">
     <xsl:call-template
       name="transformAttribute" />
    </xsl:for-each>

  </UML:Classifier.feature>
  </UML:Class>
</xsl:template>
```

The ERD Entity Properties, which are stereotyped attributes, are transformed in the similar manner by the template "transformAttribute", which is not presented in the paper.

The resulting representation after the transformation should be the same in case of tool dependent and tool independent techniques. To make it comparable, the transformed XMI version of the model was imported into Rational Rose, and the transformed entity – now as a table – was added to a new class diagram. It is presented in Fig. 3.

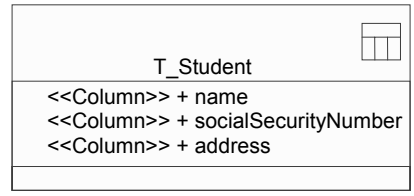

Fig. 3. Table generated for the entity shown in Fig. 2.

The graphical representation depends on the UML tool that is used to present the element. XMI documents contain no information of UML diagrams and the graphical representation of the elements. If the tool does not support the graphical representation of the stereotype is rendered as a guillement, illustrated in Fig. 4.

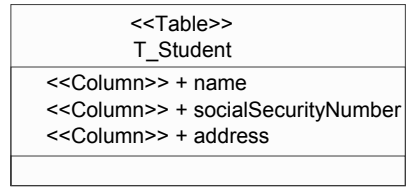

Fig. 4. Textual representation of the <<Table>> stereotype.

Although the example presented is visualized in both cases with Rational Rose, the UML tool independent approach does not restrict the usage of the UML tool. However, there is only one requirement from the tools, which is the support for the XMI model interchange format.

4.3. Remarks on implementation

In case of the entire ERD profile, there are certain limitations in Rational Rose. For example some restrictions which are defined in the profile cannot be implemented in the tool, because the metamodel of the tool does not have the same structure as the original UML metamodel. The details of these problems are discussed in [9].

There is an issue of the placement of the tool independent transformations of models in general. Although the execution of a transformation should not be bound to any UML modeling tool, results should be suitable for import into the tools. Two solutions to this problem are possible. One is to include the transformation engine into the UML tool itself (provided that the tool allows it); the other is to export the model, transform it and import it again. The difference is in the placement of a component responsible for transformation. In the first case, the component is located in the UML tool, which makes the transformation available from the UML tool itself. In the second case, the developer should leave the UML tool to perform the transformation, and return to it after the transformation is completed (exporting and importing the model manually).

5. Related work

An approach to model transformations based on XMI was already presented by Kovse and Härder [22], but it was built on a specific XMI feature, which is still not supported in a majority of UML tools – the XMI model difference tag. The tag is used to designate differences between two models during their import and export from UML tools. Furthermore, it uses a proprietary UML model repository and it assumes that the repository recognized the XMI difference tag. Even in this case only some cases of the UML model transformations could be performed.

The translation of the knowledge contained in ontologies, which was performed in the same manner is used to customize the language by providing model libraries [23, 24] and although it is not tightly connected to the issue of introducing stereotypes into UML software development, it provides an alternative usage of the same platform for model processing, which proved itself to be useful.

A modified technique is used in our research for code generation based on stereotypes in a tool independent way. The modification is that the resulting document (c.f. Fig. 1) is not the XMI encoded UML model, but a code generated for the stereotyped class.

XSLT-based code generation technique was also presented by Sturm et al. [25], but their technique is dedicated to one specific UML tool.

6. Further work

The schema presented in section 3.1 can be extended to treat uniformly stereotypes of different types, which are elaborated in [9]. To enable such uniform treatment of different types of stereotypes, the stereotype must be expressed in general XML instead of its dialect - XSLT. This definition also should contain parts that are compatible with XMI. An additional requirement is that, as the XSLT encoded stereotype definition is needed (presented in grey in Fig. 1 and Fig. 5) it must be produced from the XML definition of stereotype. To produce it, the XSLT document defining the production rules must be developed. The process of producing the XSLT encoded stereotype is presented in Fig. 5.

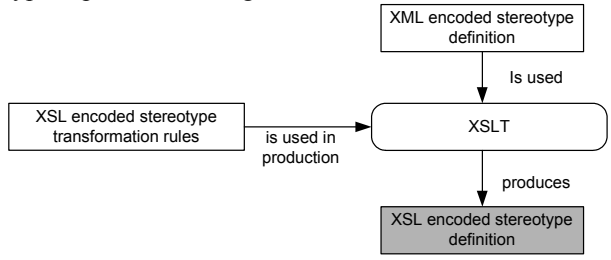

Fig. 5. Producing an XSLT encoded stereotype from an XML defined stereotype

The improvement in comparison to the previous schema is that all stereotypes belonging to a certain type need only one XSLT stylesheet for production of the

XSLT encoded stereotype definition (in grey). Furthermore, there is no need to incorporate knowledge about the XSLT language to develop each stereotype. On the other hand additional effort is needed to define the format of the XML dialect for defining stereotypes. This format can be a lot simpler than the format of XSLT. Of course there is a trade-off. The simplicity of the stereotype definition is replaced by the complexity of the XSLT stylesheet with stereotype transformation rules. The trade-off could be acceptable provided that the stereotypes used in software development processes are not divided into too many classes.

7. Summary

The paper presents an alternative way of using stereotypes in UML based designs. Instead of introducing stereotypes in UML tools directly, which is in many ways restrictive, this paper proposes a UML tool independent technique. The technique overcomes several problems with introducing and using the stereotypes in UML tools.

The advantage of the proposed method is the XSLT processing paradigm, which is based on pattern-matching. In practice it means that the Stylesheet changes only the elements that are matched with a specific XSLT expression, but leaves the other elements untouched. Therefore, the structure of the XMI document remains the same after the transformation, for example leaving the UML tool vendor specific extensions to the XMI unaltered. It must be stated explicitly that even though the tool independent stereotype introduction method may be used in a special case to translate between two XMI dialects, it is not designed for this purpose and it has not been tested for that.

References

[1] Object Management Group, "Unified Modeling Language Specification V. 1.5", Object Management Group, 2003, www.omg.org, last accessed 2003-10-01.

[2] Wirfs-Brock R., "Stereotyping: A Technique for Characterizing Objects and Their Interactions", *Object Magazine*, vol. 3, 1993, pp. 50-3.

[3] Wirfs-Brock R., Wilkerson B., and Wiener L., "Responsibility-Driven Design: Adding to Your Conceptual Toolkit", *ROAD*, vol. 2, 1994, pp. 27-34.

[4] Atkinson C., Kuhne T., and Henderson-Sellers B., "To Meta or Not to Meta - That Is the Question", *Journal of Object-Oriented Programming*, vol. 13, 2000, pp. 32-5.

[5] Henderson-Sellers B., "The Use of Subtypes and Stereotypes in the UML Model", *Journal of Database Management*, vol. 13, 2002, pp. 43-50.

[6] Kuzniarz L. and Ratajski J., "Code Generation Based on a Specific Stereotype", In the Proceedings of Information Systems Modeling, Roznov, Chech Republic, 2002, pp. 119-128.

[7] Kuzniarz L. and Staron M., "Customisation of Unified Modeling Language for Logical Database Modeling", Blekinge Institute of Technology, Ronneby, Research Report 2002:12, 2002.

[8] Gogolla M. and Henderson-Sellers B., "Analysis of UML Stereotypes within the UML Metamodel", In the Proceedings of ""UML"" 2002 - Unified Modeling Language. Model Engineering, Concepts, and Tools. 5th International Conference. Proceedings, 30 Sept.-4 Oct. 2002, Dresden, Germany, 2002, pp. 84-99.
[9] Kuzniarz L. and Staron M., "On Practical Usage of Stereotypes in UML-Based Software Development", In the Proceedings of Forum on Design and Specification Languages, Marseille, 2002, pp.
[10] Kuzniarz L. and Staron M., "On Model Transformations in UML-Based Software Development Process", In the Proceedings of Software Enginnering and Applications, Marina del Rey, CA, 2003, pp.
[11] Kay M. R., *Xslt Programmer's Reference*, 2nd ed. Birmingham, UK, Wrox Press, 2001.
[12] IBM/Rational, "Rational Rose", 2001, http://www.rational.com.
[13] Telelogic, "Telelogic Tau G2", 2002, http://www.telelogic.com.
[14] ArgoUML, "Argo UML", Tigris.org: Open Source Software Engineering, 1998, http://argouml.tigris.org/, last accessed 2003-10-06.
[15] Systemwire, "Xlinkit", Sytemwire, 2002, http://www.xlinkit.com, last accessed
[16] Gryce C., Finkelstein A., and Nentwich C., "Lightweight Checking for UML Based Software Development", In the Proceedings of Workshop on Consistency Problems in UML-based Software Development., Part of the <<UML>> 2002 conference,Dresden, Germany, 2002, pp. 124-133.
[17] Ambler S., "The Design of Robust Peristence Layer for Relational Databases", Ambysoft, 2000, www.ambysoft.com, last accessed 2003-09-30.
[18] Ambler S., "Data Modeling 101", Agile Data, 2002-2003, www.agiledata.org, last accessed 2003-02-20.
[19] Blaha M. and Premerlani W., "Using UML to Design Database Applications", *Rose Architect*, April 1999.
[20] Gornik D., "UML Data Modeling Profile", Rational Corp., Whitepaper TP162 05/02, 2002.
[21] Connolly T. and Begg C., *Database Systems. A Practical Approach to Design, Implementation and Management*, 3rd ed. Harlow, Essex, UK, Addison-Wesley, 2002.
[22] Kovse J. and Härder T., "Generic Xmi-Based UML Model Transformations", In the Proceedings of Object Oriented Information Systems, Montpellier, 2002, pp.
[23] Kuzniarz L. and Staron M., "Generating Domain Models from Ontologies", In the Proceedings of OOIS, Montpellier, 2002, pp. 160-166.
[24] Kuzniarz L. and Staron M., "Extracting Initial UML Domain Models from DAML+OIL Encoded Ontologies", In the Proceedings of PROFES, Rovaniemi, 2002, pp. 220 ff.
[25] Sturm T., von Voss J., and Boger M., "Generating Code from UML with Velocity Templates", In the Proceedings of The Fifth International Conference an The Unified Modeling Language, Dresden, Germany, 2002, pp. 150-161.

Paper III

An Empirical Study on Using Stereotypes to Improve Understanding of UML Models

An Empirical Study on Using Stereotypes to Improve Understanding of UML Models

Ludwik Kuzniarz, Miroslaw Staron, Claes Wohlin

Department of Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520, Soft Center
SE-372 25 Ronneby
(Ludwik.Kuzniarz, Miroslaw.Staron, Claes.Wohlin)@bth.se

Stereotypes were introduced into the Unified Modeling Language (UML) to provide means of customizing this visual, general purpose, object-oriented modeling language, for its usage in specific application domains. The primary purpose of stereotypes is to brand an existing model element with a specific semantics. In addition, stereotypes can also be used as notational shorthand. The paper elaborates on this role of stereotypes from the perspective of UML, clarifies the role and describes a controlled experiment aimed at evaluation of the role – in the context of model understanding. The results of the experiment support the claim that stereotypes play a significant role in comprehension of models and show the size of the improvement.

1. Introduction

The Unified Modeling Language (UML, [1]) is a general purpose visual object-oriented modeling language, which has been gaining popularity in the last decade and became de facto standard for modeling artifacts produced during object oriented software development. The language was designed in such a way that there is a well defined set of general purpose model elements and there are mechanisms, called extension mechanisms, which allow for its customization according to local needs and requirements such as a specific domain, specific software development process or specific problem. One such mechanism is the notion of a stereotype. The idea behind a stereotype in UML is that it allows branding an existing element with specific properties. The intention was that the specific properties should express some specific semantics associated with the branded model element. And this is how the stereotypes have been commonly used. But it seems that the notion can not only be used to express properties of model elements that are beyond the core semantics but also to introduce new virtual modeling elements which could improve quality properties of the models. This role of stereotypes in UML was indicated in [2] and is still not well investigated, although it reflects the original intent of introducing stereotypes into object-oriented software development in [3]. This paper is a contribution to the evaluation of the role of UML stereotypes, which are dedicated for simplification of models and improving understanding of the UML encoded development models. It

presents an empirical experiment designed to evaluate the influence of such stereotypes on the understanding of UML models. The results presented show the extent to which the stereotypes could help in improving the comprehension of UML models. The empirical study follows a similar approach to other empirical studies in software engineering, object-orientation and UML [4, 5]. The study was done on student subjects but its results can be generalized to a broader population, since the students represent a sample, which is expected to have the smallest improvement.

The structure of the paper is as follows. Firstly, the role of the specific kind of stereotypes is described in more detail in section 2. Then the design of an experiment aimed at evaluation of the role of stereotypes is presented in sections 3 and 4. Finally, the results of the experiment are presented in chapter 5, followed by a discussion of results and scale of improvement, conclusions and indications for further works in the last section.

2. Roles of stereotypes in UML

As defined in the UML specification documents [1], the main idea behind using stereotypes is to introduce new semantics to the existing model elements. The UML definition of stereotypes involves the definitions of other extension mechanisms – tagged values and constraints (how they are involved is analyzed in [6, 7]). Such a definition of stereotypes is useful for their automatic processing in UML tools, because they separate the definition of syntactical (tagged values) and semantic information (constraints). It also allows extending the language in a way, which is consistent with the definition of the language. Stereotypes are useful in automatic model transformations, like for instance code generation for a specific purpose (i.e. [8-10]).

Stereotypes (and the new semantics expressed by them) are very important if they form profiles, which are closed sets of stereotypes definitions (along with constraints and tagged values). Profiles provide a way of grouping stereotypes according to their purpose, allowing using UML for other, more specific needs (i.e. changing the language so that it is as some other notation, for instance Entity Relationship Diagrams [7]), whereas the separate stereotypes which do not constitute profiles change the separate model elements only. The most recognized profiles are the UML profile for business modeling (part of the UML specification [1]), the UML profile for scheduling and performance [11], the UML profile for CORBA [12] and the data modeling profile [13].

There is also another way of perceiving stereotypes. They provide a secondary classification of model elements. This concept was initially introduced in [14], and discussed in detail in [2]. Such stereotypes provide a means of expressing some classification of the stereotyped model elements, adding properties, which cannot be defined for all model elements of the same kind, but only for some. This kind of stereotypes can be called *model simplification stereotypes* [7], since they are intended to make models less complicated, not always involving the definition of a new semantics. Such usage of stereotypes can help readers of the stereotyped model to understand it better. These stereotypes can also be classified as *transitive stereotypes*

(according to the classification presented in [2]), because they are added to classifiers on the model level, but should also be recognized on the instance level. They are useful as a secondary classification mechanism ([15]) since they both brand the classifier and its instances with additional meaning. An example of such a stereotype (taken also from the empirical study presented in forthcoming sections) is shown in fig 1. The stereotype name is sender and it (in brief) means that instances of classes stereotyped as sender are only able to send signals (telecommunication signals – see section 3.1) to instances of other classes, but cannot receive signals from other instances. In this sense, the stereotype is attached to a classifier (a class), but its meaning and restrictions apply to the instances of this class. This explains the reason why the graphical representation is attached to both the class and its instance (see [2] for further discussions on such application of stereotypes).

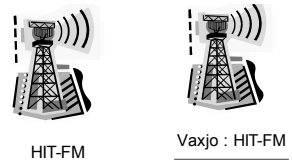

Fig. 1. Example of a transitive stereotype. The sender stereotype is applied to a class (left-hand side), while its restrictions apply to instances

Transitive stereotypes could help to distinguish between instances of standard model elements and instances of stereotyped model elements. The distinction seems to be useful in understanding the model and finding inconsistencies in models, or logical errors. An evaluation of this role of stereotypes could reveal the extent to which they influence the understanding. This paper presents a controlled experiment where the effect of stereotypes as a means for increased understanding is evaluated.

3. Experiment design

The evaluation of the role of stereotypes in software development is done using a controlled experiment. The study presented in this paper could be summarized in the following way: "The goal of the experiment is to analyze *the comprehension of UML models* for the purpose of *evaluation of UML stereotypes* with respect to *their role in understandability of UML models* from the point of view of *the software developer* in the context of *UML domain modeling*".

The type of the experiment design is a paired comparison design [16]. The treatments are model types, with two possible values - stereotyped model and non-stereotyped model. All subjects are assigned randomly to two groups (group 1 and 2). There are two rounds in the experiment. Two different sets of artifacts are presented to every subject in each group in each round. Table 1 (see section 3.6) summarizes the artifacts and their presentation to the groups. To avoid a learning effect, the artifacts come from two different application domains (A and B). The artifacts are similar in complexity and the domains are similarly common to all subjects. The above are discussed in more detail in the following sections.

3.1. Experiment objects

The set of experimental objects consists of four artifacts, summarized below (they are fully presented in [17]):

- Set A-S: stereotyped model A and description of stereotypes used in this model
- Set B-N: non-stereotyped model B,
- Set A-N: non-stereotyped model A,
- Set B-S: stereotyped model B and description of stereotypes used in this model.

Artifact set A-x describes a domain of radio and TV transmissions. It consists of a class diagram describing different types of existing objects (for instance radio station, retransmission station, different types of antennas, etc.) and a corresponding collaboration diagram describing one of possible situations (like sending a news program across a country). Excerpts of these diagrams can be found in Appendix A and are described later in this section.

Artifact set B-x describes a domain of GSM telephony. It consists of a class diagram describing different types of existing objects (for instance mobile phone, transmission station, connection to conventional telephone network, etc) and a corresponding object diagram describing one possible situation of using the network (like making phone calls at a given time).

The best solution would be to have the same models (A) in both rounds (c.f. section 5.3), but because of the learning effect in the second (subjects could understand the model better in the second round simply because they see the model for the second time) round it cannot be done. So another set of artifacts (B) is introduced, to avoid it. All the materials were ensured to contain the same information and to be of equal complexity.

The sets of experiment objects, which use stereotypes, are based on a telecommunication profile, introduced briefly in this section. This profile is used as an example because the domain of telecommunication is intuitive (with respect to the basic concepts, gathered in the profile), although other profiles can be used in the experiment. The telecommunication profile contains stereotypes which should be seen as model simplification stereotypes (according to the classification in [7]). Considering the classification of stereotypes in [2], the stereotypes are added to the elements on the model level, but their semantics concerns also elements at the model instance level (transitive stereotypes). In this paper, the definition of the profile is only informal, while the details are omitted for the sake of simplicity of the description.

The profile introduces the following modeling elements from the telecommunication domain:

- active elements like: sender, receiver and transmitter;
- passive links between the elements: transmission lines;
- signals sent between the active elements: transmissions;
- format of the content of transmission: transmission content.

Table 1 summarizes the elements in the profile along with their description.

The constraints are expressed in the Object Constraint Language (OCL, [18]) where possible. In situations where the constraints should be defined on the meta-model level, but refer to the model instance level (for instance restriction that the

operation stereotyped Transmission should be transmitted on links which are instances of classes stereotyped Transmission Line), the constraints are expressed only in natural language. In addition to OCL, every constraint is specified in natural language to make it more easily readable. An example constraint is presented below.

SC_1: Sender cannot receive any transmission - there are no operations stereotyped Transmission defined for Sender

```
context sender
inv: extendedElement->allOperations-> exists (op:Operation|
op.stereotype.name="transmission")
```

The details of the definitions are presented in [19].

Stereotype	Base class	Constraints	Description
Sender	Class	SC_x	It represents a class, which instances send telecommunication signals to instances of other classes, stereotyped Receiver or Transmitter
Receiver	Class	RC_x	This stereotype represents a class, which instances receive telecommunication signals from instances of other classes, stereotyped Sender or Transmitter.
Transmitter	Class	TRC_x	It represents a class, which instances are used to relay telecommunication signals.
Trans-mission Line	Association	TLC_x	This stereotype represents a transmission line, which allows communication between instances of stereotyped classes. It allows the transmission of telecommunication signals
Trans-mission	Operation	TOC_x	It depicts a telecommunication signal transmitted between instances of stereotyped classes.
Trans-mission Content	Classifier	TCC_x	It represents the format of information being sent as a communication signal.

Table 1. Simplified stereotype definition table for the telecommunication profile

Although the stereotypes are specified according to the UML specification, their intention is primarily to improve the understanding the UML models in which they are used, therefore they were presented to subjects in a simplified form, which contained the graphical representation and the description of the stereotype as presented in Table 1.

Excerpts of two models (A-S and A-N) can be found in Appendix A. Each model contains two diagrams (collaboration and class diagrams) and they are presented in figures 6-9. Inheritance hierarchies and notes describing the intent and restrictions of the element substituted stereotypes used in the stereotyped models. The notes were attached to classes, since classes are usually regarded as definitions of objects. The description in the notes, however, applied to objects. The reason for not attaching the notes to objects was that it would explicitly distinguish objects of different types from

each other and in that sense it would not be different than stereotyping. And such a situation was identified as one of the threats to validity of the study. It can be observed that the stereotyped collaboration diagram provides more information about the intent of the objects and therefore are more readable. It should reflect the results of the study.

3.2. Subjects

The study was carried out using software engineering (and related programs) students. It was a desired sample of the population, as is explained later in this section. The students were taking part in an object-oriented software development course, which consisted of theoretical lectures, practical exercises and individual projects. Stereotypes were not introduced during the course. There are four main potential cases (types of subjects), summarized as follows:

- A subject has worse knowledge of UML than knowledge of the telecommunication domain,
- A subject has an equal knowledge on UML and the telecommunication domain,
- A subject has better knowledge of UML than knowledge of the telecommunication domain, and
- A subject has no knowledge of either UML or the telecommunication domain.

The third type of subject is the worst-case situation. Given the knowledge of UML the subject should be able to better understand the model in standard UML (since this is the notation the subject is used to) than the model with stereotypes (since this is a new concept, which the subject is not used to). This is the situation in which the introduction of stereotypes could deteriorate the comprehension of UML models. The introduction of stereotypes in this case requires an additional effort to learn them, whereas this effort is not required when standard UML is used. This could have a negative effect on the introduction of stereotypes. Other kinds of subjects could be either expected to perform better for stereotyped models (type 1) or at least equally well (type 2 and 4).

In the study students were chosen as subjects. Since they were taught UML on a course, the knowledge of UML was sufficient to understand the given non-stereotyped model. In addition, they were not very familiar with the telecommunication domain. Therefore additional effort was required from them for understanding the stereotypes for this domain. This means that the subjects in this study primarily come from the third type above.

This group of subjects is expected to achieve the smallest improvement from all of the other groups. Therefore, the experiment can also be generalized to professionals, for who the scale of the improvement should be larger than for the sample in this experiment. Some indications on the differences between student subjects and professionals are also given in [20, 21].

3.3. Independent and dependent variables

There is one independent variable in the experiment, the diagram type, with values: S (stereotyped) and N (non-stereotyped)

Understandability of the designs is measured by two dependent variables. The variables are:

I. Total score (NRESP) – the number of correct answers for each subject when asked questions about the design

II. Time (TSEC) – the time (in seconds) which was required to fill in the questionnaire.

The type of system could be considered as a second independent variable, but it was introduced only to minimize the learning effect in the second round of the experiment, and therefore it is not an independent variable.

3.4. Hypotheses

The experiment tests the null hypothesis. If it is falsified, it would mean that the introduction of stereotypes improves the understanding of UML models. Hypotheses are formulated as follows:

- **Null hypothesis (H_0):** Introduction of stereotypes does not influence understandability of UML models.
- **Alternative hypothesis (H_1):** Introduction of stereotypes improves understandability of UML models;

3.5. Instrumentation

The main instruments used in the experiment are two comprehension questionnaires that measure the level of understanding of the presented UML models (one for each round). There are 12 questions in each questionnaire. The questions in the questionnaire concerning the same system (A or B) are identical, regardless of whether the model is stereotyped or non-stereotyped. There were three types of questions asked in each questionnaire:

1. asking for the number of instances of classes of a certain type (sender, receiver or transmitter);
2. asking for the number of different types of elements in the diagram; and
3. checking whether some elements were placed correctly according to their definition.

The questions allow measuring the level of understanding of UML models in terms of correct answers. An example of the first type of question is “How many receivers are shown on the object diagram A-S?”, which requires the subject to count objects that are either stereotyped “receiver” or inherit from the receiver class (in case of non-stereotyped model). The second type of question concerned the class diagrams, to attract the attention of subjects to the definitions of objects and to enable them to get accustomed with the class diagram. A sample question is “How many types

(kinds) of transmitters are shown in class diagram A-S?". The original diagrams contain more than one type of each element depicted by inheritance. An example of the third kind of question is "A signal cannot be transmitted via more than 2 transmitters; otherwise it is too weak to be received. How many too weak signals are shown in object diagram A-S?". It is aimed at checking the correctness of the model, i.e. an inspection-like question.

Subjects are asked to write down the time before starting answering the questions and after completing the questionnaire. The current time is displayed on the wall using a beamer. The measured total time for answering the questionnaire allows for measuring the understanding of the model in terms of the required time to answer the questions concerning the model.

After the experiment, there is a post-experiment phase, where the subjects are asked to fill in the third, additional questionnaire about their background, prior knowledge of UML, prior knowledge of stereotypes and experience in the fields of software development as well as object-orientation. This is required to check that the subjects belong to the desired population.

3.6. Experiment operation

Before the main experiment phase, there is a pilot study, which should be operated in a similar way as the main experiment. However, its intention is to validate the experiment objects and identify potential confounding factors of the study.

In the course of the main experiment, there are two rounds. In each round, each of the two groups is given a different treatment. Table 2 presents the outline of the experiment operation. The artifacts sets presented in the table are described in detail in Section 3.1. In each round each subject gets a different type of artifact set (as presented in Table 2).

	Round 1	Round 2
Group 1	Set A-S	Set B-N
Group 2	Set A-N	Set B-S

Table 2. Experiment rounds

The design of the experiment includes a lecture, given directly before the experiment. The lecture explains the notion of stereotypes and showing some basic examples of the usage of stereotypes, but it is not meant to introduce the set of stereotypes used in the experiment.

4. Conducting the study

The study was performed in two steps. The first step was a pilot study to examine the context of the study and to determine some of possible confounding factors that could influence the results of the study. The second step was the experiment, which was

aimed at hypothesis testing. The results of the pilot study identified a confounding factor, which caused a change of instrumentation in the experiment (as presented in Section 5.1).

The experiment was conducted in approximately 2 hours on a sample of 44 students. At the start, the subjects were given a 45 minutes lecture introducing the notion of stereotypes, explaining the usage of stereotypes and its graphical representation. The telecommunication profile was not explained during the lecture. Then, the subjects were divided into two equal groups using blocking. The blocking was done based on the study program of the students and the laboratory group. Both groups were in the same room as the lecture. Then, the subjects were given a short introduction to their task. The time was displayed on the projector during the whole time of experiment. The subjects were given the first comprehension questionnaire (with stereotyped or non-stereotyped model with respect to the group they belonged to). After completing the first comprehension questionnaire the subjects were given the second comprehension questionnaire and after completing the second one, they were given the background questionnaire to fill in.

5. Experiment results

The results of the experiment indicate that the alternative hypothesis H_1 can be supported and that the null hypothesis can be rejected. The results are presented in section 5.2, but some interesting findings, which influenced the main experiment, are taken from the pilot study as described in section 5.1.

5.1. Pilot study results

The pilot study was done on a group of two subjects, who were chosen based on their knowledge of UML and telecommunication domain (see section 3.2). Each of them was acting as one group (as described in Table 2). The results from the pilot study showed that there existed a confounding factor in the study. For a subject who was given the stereotyped model in the first round, the time for solving the test in the second round was shorter than in the first round. It was because the subject used some of his knowledge about stereotypes (from round 1) to introduce the stereotypes to the non-stereotyped model in the second round. This introduction helped the subject to improve the time for solving the assignment – questionnaire (since the questions in both comprehension questionnaires were of the same kind) and the number of correct answers.

From the pilot study it was also found that one of the models was slightly less complicated. There was also an ordering effect in the questionnaire, which made the introduction of stereotypes in the second round easier and intuitional.

The expected result was achieved by the subject representing group 1. The opportunity of having a dialogue with the subject also proved that the subject perceived stereotypes as helpful in understanding. The subject indicated that one of the models was less complex (the same as the subject representing group 2).

As a result of the study, the order of questions in the questionnaires was changed and the complexity of all models was balanced. The pilot study also indicated the extent to which the introduction of stereotypes could be useful. By introducing the stereotypes in the second round, the subject showed that the set of stereotypes was indeed helpful in understanding the model.

5.2. Experiment results analysis

The experiment was performed on a group of 44 students. After the initial data set reduction it was found that answers from 39 (20 in group 1 and 19 in group 2) subjects could be taken into consideration. Five subjects were removed due to some errors in given documents (two subjects solved two tests for the same system) and personal issues (not handing in one of the questionnaires). The results of the experiment indicate that introduction of stereotypes improves the understandability of UML models. It is interesting to examine the results from different perspectives: each variable (NRESP and TSEC) independently, relative times for a correct answer (TSEC/NRESP) and overall subject improvement. Each variable (TSEC, NRESP and TSEC/NRESP) was tested for the normal distribution using the Chi-2 test. And since the test indicated that none of the distributions could be classified as normal, a non-parametric test (Wilcoxon) was used to analyze the data. The test is recommended for this type of design by for instance [16], and its detailed description could be found in [22]. For each analyzed variable a bar plot is used as a presentation of the acquired data.

5.2.1. Number of correct responses

One of the two direct measures – the number of correct responses (NRESP) – allows judging how the introduction of stereotypes influences the understanding of models in terms of accuracy. The influence of stereotypes on the number of correct responses for each subject is summarized in Fig 2.

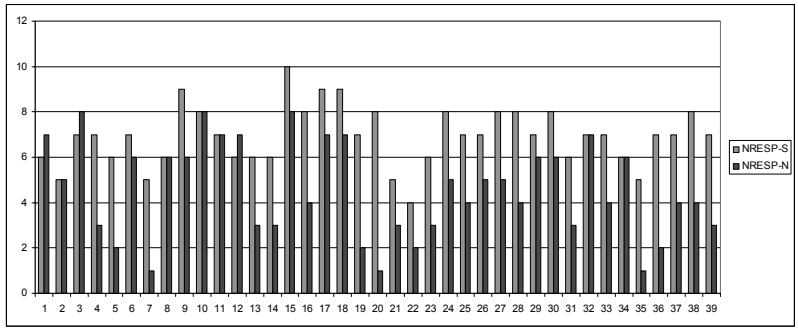

Fig. 2. Number of correct answers for each subject. NRESP-S is the number of correct answers for the stereotyped model and NRESP-N is the number of correct answers in the non-stereotyped model.

The basic descriptive statistics indicated that the null hypotheses could be rejected. The results of the analysis are summarized in Table 3.

Treatment	Measure	Value	Percentages
NRESP-S	Mean	6.92	152%
	Standard deviation	1.29	
NRESP-N	Mean	4.56	100%
	Standard deviation	2.01	
Differences NRESP-S – NRESP-N	Mean	2.36	52%
	Standard deviation	1.87	

Table 3. Summary of analysis results for NRESP variable

The higher mean value and the lower standard deviation for the stereotyped model show that the subjects understood the model better and they were more consistent in their answers when the stereotypes were involved. The improvement can be measured in percentages, taking as the 100% the mean value of the number of correct answers for the non-stereotyped model. The value of the improvement is 2.36/4.56*100%= 52%.

The analysis by the Wilcoxon test showed that the null hypothesis can be rejected in favor of the alternative hypothesis with the significance level (p) of less than 0.0001.

5.2.2. Time spent for answering the questionnaire

The analysis of the time spent for answering the comprehension questionnaire (TSEC) allows judging the influence of stereotypes on the time spent for answering the questionnaire. The times acquired from the subjects are summarized in Fig 3.

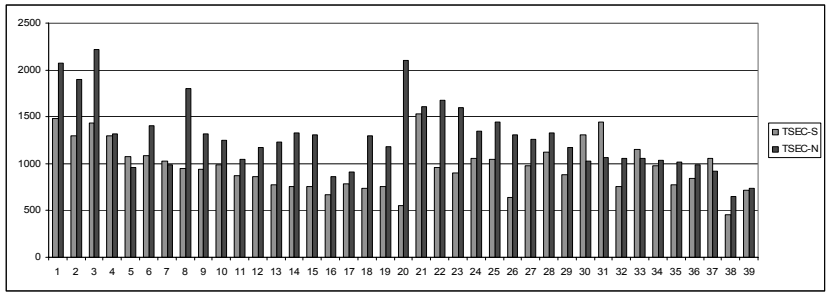

Fig. 3. Time spent (in seconds) for answering the questionnaire for each subject. TSEC-S is the time spent for the stereotyped model and TSEC-N is the time spent for the non-stereotyped model.

The analysis with descriptive statistics indicated that there was an improvement. The results of the analysis are summarized in Table 4.

Treatment	Measure	Value (sec)	Percentages
TSEC-S	Mean	967	75%
	Standard deviation	260	
TSEC-N	Mean	1281	100%
	Standard deviation	367	
Differences TSEC-S – TSEC-N	Mean	-315	25%[1]
	Standard deviation	356	

Table 4. Summary of the analysis for the TSEC variable

Since the mean value for the stereotyped model was lower than the mean value for the non-stereotyped model, the average time required understanding the model was shorter. The subjects were also more consistent for the stereotyped model (lower value of the standard deviation). It shows that on average it takes less time to understand the stereotyped model than the non-stereotyped model. It is an interesting observation, since the subjects had to spend some time to understand the stereotypes (and this time was not required for the non-stereotyped model). The average improvement was on the level of 25%. As a basis for calculations, the mean value for the non-stereotyped model was taken (1281 = 100%).

The analysis using the Wilcoxon test showed that the null hypothesis can be rejected in favor of the alternative hypothesis with the significance level (p) of less than 0.0001.

5.2.3. Relative time required for a correct answer

An important correlation between the TSEC and NRESP variables is the relative time for a correct answer (TSEC/NRESP). The comparison between the relative times

[1] In this case the absolute value (314.69) was used in the calculations.

required for a correct answer in each round for each subject gives an overview of the overall performance of a subject in each round. Fig 4 presents the summary of the results.

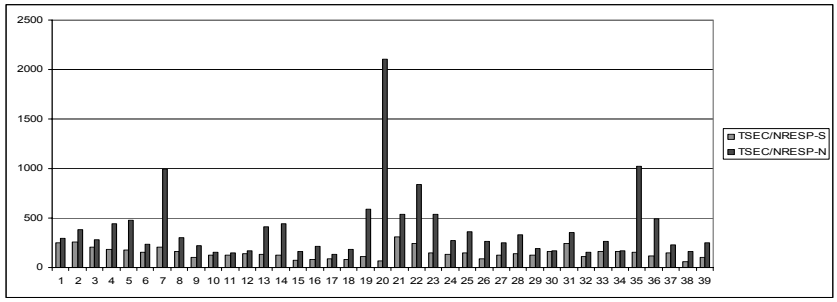

Fig. 4. Relative time (in seconds) required for a correct answer. TSEC/NRESP-S is the relative time for the stereotyped model and TSEC/NRESP-N is the relative time for the non-stereotyped model.

The results of the analysis using descriptive statistics are presented in Table 5.

Treatment	Measure	Value (sec)	Percentages
TSEC-S/NRESP-S	Mean	147	38%
	Standard deviation	55	
TSEC-N/NRESP-N	Mean	389	100%
	Standard deviation	355	
Differences TSEC-S/NRESP-S – TSEC-N/NRESP-N	Mean	-242	62%[2]
	Standard deviation	355	

Table 5. Summary of the analysis for the TSEC/NRESP variable

The mean value for the differences indicates that on average the relative amount of time for a correct answer was shorter for the stereotyped model. It means that on average the subjects were more efficient in giving the correct answer for the stereotyped model.

An interesting observation from this variable was that there were no positive differences (c.f. Fig 4). It means that the relative times for a correct answer were better for every subject for the stereotyped model. Since the mean value is also lower for the stereotyped model, then a conclusion could be drawn that on average, the time required for one correct answer was shorter in the case of the stereotyped model. It gives a good indication of improvement since the average number of correct answers was higher for the stereotyped model. The average change was on the level of 62% of

[2] In the course of calculations, the absolute value was taken.

improvement in the relative time. Again for the calculations the mean value in the non-stereotyped model was chosen as a basis (100%),

Subjects 7, 20 and 35 can be identified as outliers in the analysis for this variable, since their results are more than two standard deviations away from the mean. They were not identified as outliers in the analyses of the previous variables, because the data was more uneven distributed (larger standard deviation). It was found that the outliers had a significant influence on the analysis for the relative time. The summary table after removing the outliers is as follows:

Treatment	Measure	Value (sec)	Percentages
TSEC-S/NRESP-S	Mean	147	48%
	Standard deviation	56	
TSEC-N/NRESP-N	Mean	307	100%
	Standard deviation	157	
Differences TSEC-S/NRESP-S – TSEC-N/NRESP-N	Mean	-159	52%[3]
	Standard deviation	137	

Table 6. Summary of the analysis of the TSEC/NRESP variable after removing outliers

Removing the outliers resulted in a decrease of the improvement in this analysis compared to the results from Table 5. The improvement in this case was 52%.

The Wilcoxon test showed that the null hypothesis could be rejected at the significance level of less than 0.0001 for both variants of analysis – with and without outliers.

5.2.4. Overall improvements

The highly desired effect of the experiment was that there should be an improvement in both variables (number of correct responses and time) or in either of them (without influencing the other one). A strongly undesired effect would be that there is actually deterioration for both variables, i.e. fewer correct answers in a longer time. Between the extreme cases, there are results where one variable is improved and the other is deteriorated. An analysis of the effect when one of the variables was improved and the other is deteriorated must be done in the context of analysis of a relative time for a correct answer and the analysis of separate variables. Since the results indicated that the relative time was better (for each subject) for the stereotyped models (c.f. previous section), they may be regarded as an overall improvement. The results of the study from the overall improvement perspective are summarized in Figure 5.

[3] In the course of calculations, the absolute value was taken.

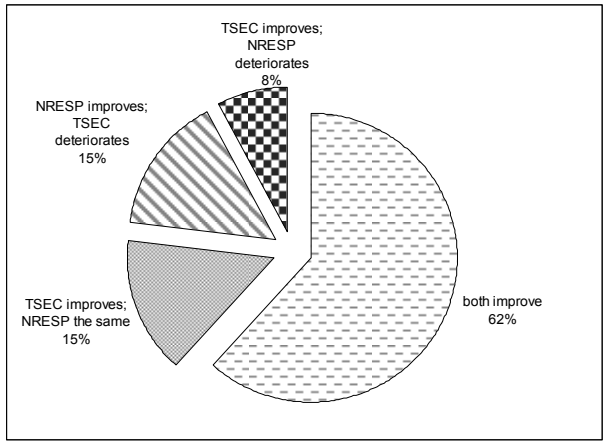

Fig. 5. Overall improvement chart

The chart indicates that the improvement both in terms of time and number of responses was achieved by 62% of the subjects. Some kind of improvement (including improvements of only one variable) was achieved by 77% (62% + 15%). The improvement of only one variable (while deterioration of the other) was achieved by 23% of subjects. The improvement in at least one of the variables (not counting deterioration in the other) was achieved by 100% of the subjects (62% + 15% + 15% + 8%). There was no situation where the deterioration was achieved in both variables. The chart shows that in the majority, the introduction of stereotypes improved the performance of subjects (77%). The remaining 23% must be analyzed in the context of a relative time for a correct answer analysis – TSEC/NRESP. Because there was no deterioration in relative time for any subject, a conclusion can be drawn that the deterioration in the absolute time values were compensated by the number of correct answers (which were much higher for the stereotyped model). In this context, the 23% percent of the subjects, who achieved improvement for one variable and deterioration in the other, achieved the overall improvement. There were no other possible situations, i.e.:

- deterioration in both variables,
- NRESP the same and TSEC deteriorates;
- NRESP improves and TSEC stays the same; and
- TSEC stays the same and NRESP deteriorates.

5.3. Threats to validity of the study

As any empirical study, this study has threats to its validity. The threats are grouped as suggested in [16] and presented below.

One of the most important threats to construct validity is the effect of interaction of testing and treatment. In group 2, each subject was given the stereotyped model prior to the non-stereotyped model. This could result in the introduction of stereotypes in the second round (for non-stereotyped model). Since it was indicated by the pilot

study, the ordering of questions was such that it minimized the effect and the models were prepared in a way, that introduction of stereotypes required some effort, which can be seen in the analysis of times for solving the test.

There is also a conclusion validity threat. Since the experimenters prepared the objects, there is a danger that the complexity of the design documents is not the same as the complexity of real-world design documents. On the other hand, the prior preparation of the objects resulted in an equal complexity of the models (which was seen as a larger threat).

The main external validity threat is that the sample may be considered as too homogenous, since it consists of students of the same year. This in a sense is a desired effect. Since the students represent a group of subjects that has the most undesired background for the evaluation (as described in section 3.2). The results of the background questionnaire showed that the subjects know UML in practical applications, at the same time they are not familiar with the domain at the same level.
The design of the study was done in a way to minimize the threats to the internal validity of the study (although sometimes introducing the other kinds of threats as discussed above), and thus there are no major threats in this category.

5.4. Discussion of results

Firstly, the results show clearly that introduction of stereotypes improves the understandability of UML models in terms of time required to answer the comprehension questionnaires for each model and the number of correct answers in them. Naturally there were some exceptional cases where the results indicated the negative effect (unlike the expected), but these cases formed a small number of all cases.

Secondly, the analysis of overall performance has a large importance from the practical perspective. It showed that a great majority of the subjects achieved an improvement of some kind, without deterioration in other aspects. All subjects acquired some improvement, but some of them decreased the other aspect (other variable).

Thirdly, despite the decrease in one variable, the overall performance of these subjects was positive, since the analysis of the relative time for a correct answer showed that for every subject, the relative time was shorter for the stereotyped model than for the non-stereotyped model.

Fourthly, the results showed a complete lack of the highly undesired effect of deterioration of the performance of the subject in terms of both variables. Although the sample was highly selective, the lack of such effect indicates that in the real-world environment its influence will most likely have the positive effect too.
Finally, the results show that the benefits of using the stereotypes when it comes to understandability are obvious. However, the costs of introduction of stereotypes (including their proper definition, introduction into tools, etc) are not studied. It requires further analysis.

6. Conclusions and further work

The empirical study of the role of stereotypes in understanding of UML models shows clearly that the stereotypes improve understandability. The improvement was achieved in the following three aspects. Firstly, an important measure is the number of correct answers in the tests checking the level of understanding. The improvement achieved in this category was 52%. Secondly, the improvement in time required for solving the test was on the level of 25%. Finally, the improvement in a relative time for a correct answer was achieved on a level of 62%. A notable fact is that an improvement of some kind was achieved by every subject in the study. Since the sample chosen for the experiment was a representative of the worst-case scenario sample, the results for professionals (who do not belong to the worst-case scenario group) are expected to be even more positive. Their background (knowledge of UML but not of the domain) means that the introduction of stereotypes could deteriorate the understanding of the models (because of the introduction of new, previously unknown elements).

The study presented in the paper is an introductory study. The further research in this field should be a case study on usage of stereotypes in their natural context, for instance a specific domain or specific software development process. Furthermore, the evaluation of the scale of to which stereotypes help in understanding the models in the industrial applications can be done by a case study.

References

[1] Object Management Group O., *Unified Modeling Language Specification V. 1.4*, 2002.
[2] Atkinson C., Kuhne T., and Henderson-Sellers B., "Stereotypical Encounters of the Third Kind", In the Proceedings of UML 2002, Dresden, 2002, pp. 100-114.
[3] Wirfs-Brock R., Wilkerson B., and Wiener L., "Responsibility-Driven Design: Adding to Your Conceptual Toolkit", *ROAD*, vol. 2, 1994, pp. 27-34.
[4] Otero M. C. and Dolado J. J., "An Initial Experimental Assessment of Dynamic Modeling in UML", *Empirical Software Engineering*, vol. 7, 2002, pp. 27-37.
[5] Briand L. C., et al, "An Experimental Comparison of the Maintainability of Object-Oriented and Structured Design Documents", *Empirical Software Engineering*, vol. 2, 1997, pp. 291-312.
[6] Gogolla M. and Henderson-Sellers B., "Analysis of UML Stereotypes in the UML Metamodel", In the Proceedings of UML 2002, Dresden, 2002, pp. 84-99.
[7] Kuzniarz L. and Staron M., "On Practical Usage of Stereotypes in UML-Based Software Development", In the Proceedings of Forum on Design and Specification Languages, Marseille, 2002, pp.
[8] Uhl A. and Lichter H., "A UML Variant for Modeling System Searchability", In the Proceedings of Object Oriented Information Systems, Monpellier, 2002, pp. 199-211.

[9] Kuzniarz L. and Ratajski J., "Code Generation Based on a Specific Stereotype", In the Proceedings of Information Systems Modeling, Roznov, Chech Republic, 2002, pp. 119-128.
[10] Rational, "Rose Data Modeling Profile", Rational, 1999, www.rational.com, last accessed 2002-10-01.
[11] Object Management Group, "UML Profile for Schedulability, Performance and Time", Object Management Group, 2002, www.omg.org, last accessed 2003-09-20.
[12] Object Management Group, "UML Profile for Corba", Object Management Group, 2002, www.omg.org, last accessed 2003-09-20.
[13] Gornik D., "UML Data Modeling Profile", Rational Corp., Whitepaper TP162 05/02, 2002.
[14] Wirfs-Brooks R., Wilkerson B., and Wiener L., "Responsibility-Driven Design: Adding to Your Conceptual Toolkit", *ROAD*, vol. 2, 2003, pp. 27-34.
[15] Wirfs-Brock R., "Stereotyping: A Technique for Characterizing Objects and Their Interactions", *Object Magazine*, vol. 3, 1993, pp. 50-3.
[16] Wohlin C., Runeson P., Host M., Ohlsson M. C., Regnell B., and Wesslen A., *Experimentation in Software Engineering: An Introduction*. Boston MA, Kluwer Academic Publisher, 2000.
[17] Staron M., "Experiment on the Role of Stereotypes in UML Based Software Development - Materials", 2003, http://www.ipd.bth.se/mst/Experiment/index.html, last accessed 2003-09-03.
[18] Object Management Group, "Object Constraint Language (Ocl) Specification V. 1.4", Object Management Group, 2002, www.omg.org, last accessed 2003-01-10.
[19] Kuzniarz L., Staron M., and Wohlin C., "An Experimental Evaluation of the Role of Stereotypes in Understanding of UML Models", to be published as Technical report, Blekinge Instiute of Technology, Ronneby, Technical report Technical report, 2003.
[20] Höst M., Regnell B., and Wohlin C., "Using Students as Subjects - a Comparative Study of Students and Professionals in Lead-Time Impact Assessment", *Empirical Software Engineering*, vol. 5, 2000, pp. 201-214.
[21] Tichy W., "Hints for Reviewing Empirical Work in Software Engineering", *Empirical Software Engineering*, vol. 5, 2000, pp. 309-312.
[22] Siegel S. and Castellan N. J., *Nonparametric Statistics for the Behavioral Sciences*, 2nd ed. New York, McGraw-Hill, 1988.

Appendix A – artefacts' excerpts

Sample parts of models shown to subjects in the study are presented herein to give an overview of the diagrams. For the sake of simplicity, only parts of the models are included and only from one model (A), the other model (B) was similar in the content, although it defined different elements (for instance telephones instead of antennas). All materials can be found in [16].

Artefact set A-S

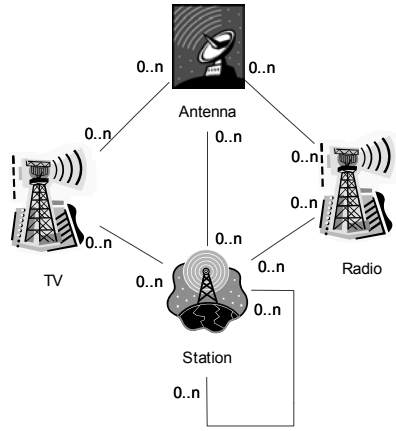

Fig. 6. Excerpt from class diagram in artefacts set A-S (about 50% of the diagram)

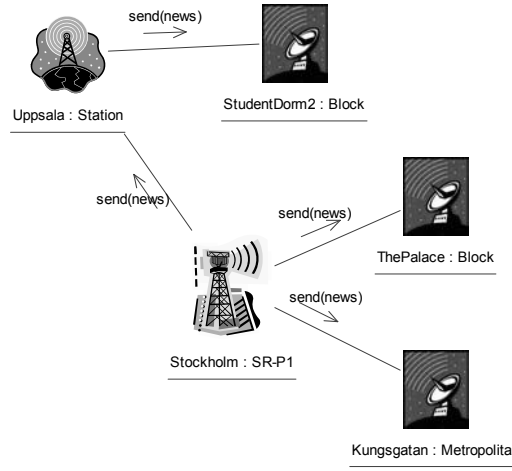

Fig. 7. Excerpt from the collaboration diagrams from artefacts set A-S (around 20% of the diagram).

Artefact set A-N

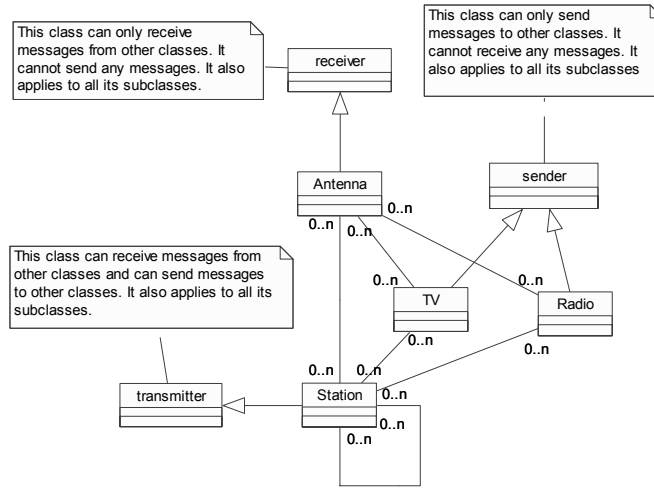

Fig. 8. Excerpt from the class diagram in artefacts set A-N (about 50% of the diagram).

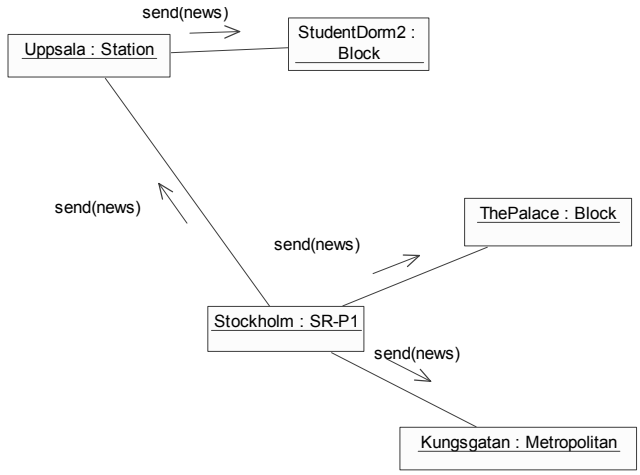

Fig. 9. Excerpt from the collaboration diagram from set A-N (about 20% of the diagram).

Paper IV

Generating Domain Models from Ontologies

Generating Domain Models from Ontologies

Ludwik Kuzniarz, Miroslaw Staron

Department of Software Engineering and Computer Science
Blekinge Institute of Technology
S-372 25 Ronneby, Sweden
(Ludwik.Kuzniarz, Miroslaw.Staron)@bth.se

The paper presents and elaborates on the idea of automatic acquisition of knowledge about domain structures from ontologies into an object-oriented software development process. The information required to be included in the domain model produced during the development process is identified. The existence of the knowledge in ontologies is investigated. Requirements for ontology description languages are formulated followed by a brief evaluation of existing languages against these requirements. A schema for domain knowledge acquisition process is outlined. A realization of the schema is sketched in the paper and illustrated with an example.

1. Introduction and motivation

A vision of using the existing information, and more general, knowledge by different users or agents for specific applications was presented in [1]. This vision was partially realized by the introduction of the Semantic Web [2, 20] where knowledge is expressed in *ontologies*. In general ontologies provide a shared and common understanding of a domain that can be communicated between people and heterogeneous and distributed application systems [13]. It is also a formal specification of a conceptualization or an abstract model for some phenomenon in the world which identifies relevant concepts of the phenomenon [14,18]. The software development process is an activity where producing an abstract model for a phenomenon in the real world called a domain model is an important issue. The domain model is constructed at a very early stage of the development process and it is usually done manually from scratch based on investigation in the problem domain [4]. But the required knowledge may already and in many cases does exist and is stored as an ontology. So the question is if this knowledge can be automatically acquired to the development process. Formally such acquisition should be a transformation from an ontology specification language to a language used to express artefacts produced throughout the development process. This paper elaborates on the general idea of automatic acquisition of the knowledge from ontologies into software development. The information used in domain models is identified, and it is shown that the information can be acquired from ontology. Requirements for ontology description language are imposed to make it possible to extract the structural knowledge about the domain from the ontology describing the domain. The most widely used languages for

describing ontologies are shown to fulfill the requirements. Based on this a schema for a domain knowledge acquisition process is outlined and its practical realization is sketched. The process is a general description of a concrete knowledge acquisition method from ontologies expressed in DAML+OIL into a domain model expressed in the Unified Modeling Language [22] presented in [11].

The paper contains a short description of domain models and ontologies in Section 2 and Section 3, which is followed by a description of a knowledge acquisition process in Section 4. Based on this an automatic knowledge acquisition process from DAML+OIL encoded ontologies into a UML domain model is sketched in Section 5. The paper ends with a short evaluation of the proposed knowledge acquisition process.

2. Domain models

A domain model is an artefact produced at the early stage of the software development process and illustrates meaningful (to the modelers) conceptual classes in a problem domain. According to [16,17] the domain model is a set of class diagrams that show:

- domain objects or conceptual classes,
- associations between conceptual classes, and
- attributes of conceptual classes.

Although there are more elements that can be expressed by languages used for domain models, the elements above are crucial for object-oriented analysis. They form the knowledge about the domain, and help the modelers to understand the problem area. One of the most widely used languages for object oriented modeling is the Unified Modeling Language (UML). It has a rich set of elements used for modeling software, but only few of them are actually used for domain models. The useful elements are classes, their attributes, associations between them, and inheritance. A sample domain model expressed in UML for a part of reality describing relationships between such concepts as Person, Man and Woman is presented in Fig. 4. The key rationale behind domain models is that they are a kind of visual dictionaries of *abstractions*. They depict abstractions of conceptual classes. Despite that the description of phenomena like man and woman is very rich, only some concepts are of interest from the given domain perspective. The domain model displays a partial view, or abstraction and ignores uninteresting (to the modelers) details. Usually identification of the meaningful abstractions is performed manually by the analyst, who identifies the concepts and relationships among them. However, the concepts may already be described, they may already be a part of the knowledge that is used in different areas. One such area is the Semantic Web, where knowledge is expressed by means of ontologies.

3. Ontologies

The term Ontology comes from philosophy, where it is used to characterize a science about the nature of things, what types of things exist and what are the relationships among them. It was adopted for computer science where it describes the concepts of some phenomenon that exists in the real world and how it is connected to other phenomena [10]. Ontologies are widely used for describing resources that exist on the Web, and for adding some semantics to these resources. They contain knowledge about domains, which can also be used by mobile agents, which need information not only about the knowledge they encounter, but also about its structure.

There are several languages used to describe ontologies. Some of them are based on first order logic and provide a programming language-like syntax (like Knowledge Interchange Format [18]). Other are dedicated for use with ontologies, like Ontology Interchange Language (OIL) [9] or dedicated for use with agents, like DARPA Agent Markup Language (DAML) [7]. The latter two are similar in the design and concept, and as they are designed to serve for describing ontologies, a unifying activity has been performed and a joint specification has been introduced for ontology description language. The language is called DAML+OIL [21]. It is a markup language based on XML. It is built on top of the Resource Description Framework (RDF) and its schema (RDFS) [19]. DAML+OIL contains elements and mechanisms which allow expressing not only the knowledge, but also its structure in the same document. The structure of the knowledge is described by a set of classes, attributes and relationships between classes. Knowledge that belongs to the described domain consists of objects that have concrete values of the attributes and are connected to other concrete objects. DAML+OIL is the emerging standard for ontology description language. It is used not only by research communities across academia, but also industry. It can express the concepts identified in Section 2, for example, a sample class defined in DAML+OIL can be expressed by the following set of tags (from [21]):

```
<daml:Class rdf:ID="Man">
   <rdfs:label>Man</rdfs:label>
   <rdfs:SubclassOf rdfs:Resource="#Person">
</daml:Class>
```

A sample ontology describing the same domain as in Section 2 expressed in DAML+OIL is visually presented in Fig. 2. Elements required for domain models are identified in Section 2. Based on that, there are some requirements that need to be imposed on language used to describe ontologies. If the requirements are fulfilled, then the domain knowledge can be acquired from the ontology. The requirements are that the language must allow to:

- express classes - concepts significant for the domain,
- express attributes of classes - properties of the concepts, and
- express relationships between classes - both is-kind-of relationships and associations between concepts.

The languages presented in this paper fulfill the above requirements. DAML+OIL is the emerging standard for the ontology description language, thus it is used as a

base for the practical realization of the domain acquisition process described in more detail in [11]

4. Domain knowledge acquisition process

Acquisition of knowledge is a translation from an ontology description language to a language used for domain modeling. Figure 1 outlines the process. Because there is more knowledge in ontologies than is needed in domain models, some of the knowledge must be filtered out. It is possible to acquire the knowledge directly from ontology (depicted by a dashed line), but it is advisable to introduce an additional, intermediate step. It is a filtration of elements of ontologies identified in Sections 2 and 3. The result of this step is an ontology, which contains only the meaningful concepts, which are transformed into the language used for domain modeling. The filtration skips the elements of ontology which are not meaningful from domain modeling perspective. If the ontology description language could express only the concepts required (identified in Section 2), then the ontology of meaningful concepts and the original ontology would be identical. But since ontology description languages can express more than it's required, then the ontology presented on gray background in Fig. 1 is a subset of the original ontology. The transition from the ontology to a domain modeling language is a translation between the languages. In case of using DAML+OIL as the ontology

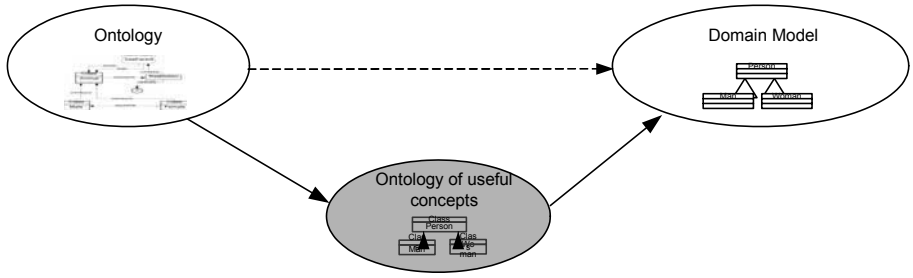

Fig. 1. Knowledge acquisition process schema

description language and UML as the language for domain modeling, the translation is a transformation. Since UML can be represented by an intermediate language, namely the Extensible Metadata Interchange (XMI) [23], which is a dialect of XML, the transformation can be performed automatically with use of technologies dedicated for XML transformations. The details of the transformation and filtration processes are described in [11].

5. Automatization of the process

It is desirable that the acquisition process is automated. The automated process and technology used for it depends on the language used for ontologies and the language used for domain modeling. If the languages are DAML+OIL and UML respectively, then the process can be performed with use of the Extensible Stylesheet Language for Transformations - XSLT [12]. The reason for using XSLT is that UML can be represented with use of XMI. Therefore, the process of acquiring the knowledge is a filtration and a subsequent translation, both performed using XSLT. The filtration is an XSLT transformation from a DAML+OIL encoded ontology into a DAML+OIL encoded ontology with only meaningful concepts. The translation is an XSLT transformation, which changes the representation of the same concepts from a DAML+OIL encoded ontology to an XMI encoded UML domain model. The details of both the filtration and the transformation are described in [11]. As DAML+OIL is built on top of RDF and RDFS, the translation is described only for those elements that are expressed with use of DAML+OIL. The relationship and mapping between RDFS constructs and UML are presented in [4].

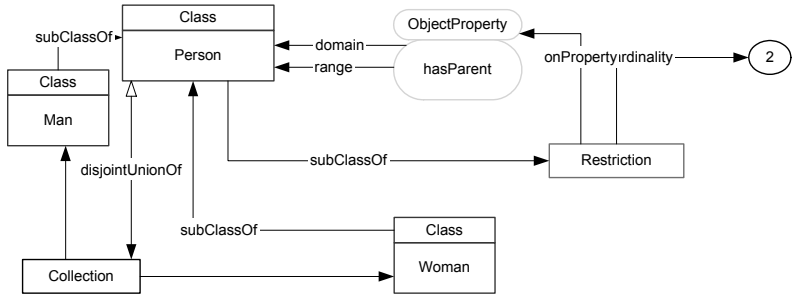

Fig. 2. Domain knowledge expressed as DAML+OIL encoded ontology

To illustrate how the acquisition process can be performed in practice, a small example is presented. It shows how the knowledge about a domain is introduced into a domain model. DAML+OIL is used to describe the ontology of the domain, and UML is used to express the domain model. An ontology describing the domain is presented in Fig. 2.

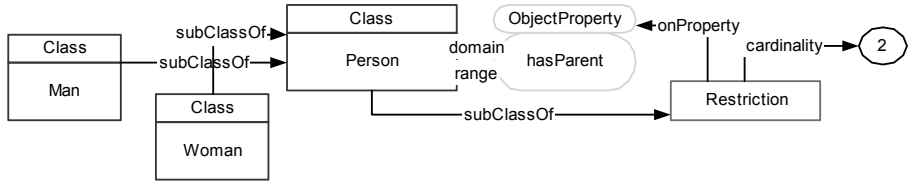

Fig. 3. DAML+OIL encoded filtered ontology

The knowledge contains such concepts as Person, Man and Woman. It also describes a relationship hasParent, between two instances of class Person and shows

that class Man is a subclass of class Person (is-kind-of relationship). The ontology also contains information that is not required from the domain model perspective, which is the information that each instance of class Person is either a man or a woman (disjointUnionOf relationship in Fig. 2). Since the DAML+OIL encoded ontology contains some additional information about the domain than is required, the filtration produces a new ontology, which contains only the meaningful information from a domain modeling perspective. The filtered ontology is presented in Fig. 3. In this example the resulting ontology does not contain the information that each person is either a man or a woman. It is semantic information, which is not relevant for domain modeling. The next step is a transformation which changes the representation from DAML+OIL to UML. The domain model, which is the result of the acquisition process is presented in Fig. 4. It contains all classes that were in the original ontology (Person, Man and Woman), and some relationships between them (the relationship hasParent and inheritance).

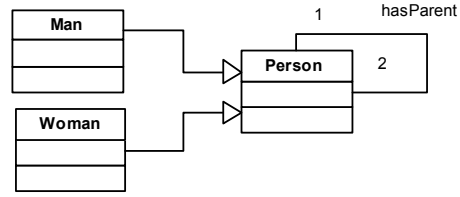

Fig. 4. Domain knowledge expressed as UML domain model

The constructed domain model is an initial model, which needs to be refined by an analyst, who should add concepts that were not in the ontology, but are important for the constructed software. In general, the domain knowledge acquisition process produces only the initial version of the domain model. The model should be then refined by analysts, who add necessary concepts, removes the obsolete ones and changes the existing concepts according to the requirements for the constructed software.

6. Conclusion

The paper deals with the general problem of knowledge sharing and reuse. It presents and elaborates on the process of automatic acquisition of domain knowledge from ontologies into domain models used in software development processes. The elements that are required for domain models were identified, and from this perspective requirements for ontology description languages were imposed. Some of the available languages were investigated. It was also shown that some of the most commonly used languages for describing ontologies fulfill the requirements. A general method of acquiring the knowledge from ontologies into software development process was described. The practical acquisition process for DAML+OIL as an ontology description language and UML as an object-oriented modeling language was outlined. The details and practical realization of the method for these languages can be found in [11]. An important feature of the acquisition process is that it ensures consistency of the obtained domain model with the existing knowledge.

As it is possible to use UML as an ontology description language [5, 6], the idea is not used in this paper. The main feature of the acquisition process presented in this paper is that is provides a mean to reuse the existing domain knowledge, thus incorporating it into newly build software.

7. References

[1] Barners-Lee T., Hendler J., Lasilla O., *The Semantic Web,* Scientific American, May 2001.

[2] Barners-Lee T., *Semantic Web Road map,* WWW Consortium September 1998, http://www.w3c.org/DesignIssues/Semantic.html.

[3] Booch G., *Unified Software Development Process,* Addison-Wesley, 1998.

[4] Chang W. W., *A Discussion of the Relationship Between RDF-Schema and UML*, W3C note, document http://www.w3.org/TR/1998/NOTE-rdf-uml-19980804.

[5] Cranefield S., *UML and the Semantic Web*, In the Proceedings The International Semantic Web Working Symposium, SWWS 2001 Symposium, Stanford 2001, pp. 113-130.

[6] Cranefield S., Purvis M., *UML as an Ontology modeling language,* in Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99), 1999.

[7] DAML language specification, http://www.daml.org/2000/10/daml-ont.html, 2000.

[8] Fenzel D., *The Semantic Web and its languages,* IEEE Intelligent Systems, November/December 2000.

[9] Fenzel D., Horrocks I., Van Harmelen F., Decker S., Erdmann M., Klein M., *OIL in a nutshell*, In the Proceedings of The 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2000), volume 1937 of Lecture Notes in Artificial Intelligence, Springer Verlag 2000, pp. 1-16.

[10] Gruber T. R., *A Translation approach to portable ontology specification*, Knowledge acquisition 5(2), October 2000.

[11] Kuzniarz L., Staron M., Hellman E. Extracting information about domain structure from DAML+OIL encoded Ontologies into UML Domain Models, Blekinge Institute of Technology, Research Report 2002:02, ISSN 1103-1581, 2002.

[12] Kay M., *XSLT, Programmer's reference,* Wrox press, 2000.

[13] Klein M., Fenzel D., Van Harmelen F., Horrocks I., *The Relation between Ontologies and Schema-languages: Translating OIL-specifications in XML-Schema*, In the Proceedings of The Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence ECAI-00, Berlin, Germany August 20-25, 2000, http://delicias.dia.fi.upm.es/WORKSHOP/ECAI00/7.pdf, accessed on 2003-10-21.

[14] Klein M., Broekstra J., Decker S., Fensel D., Harmelen F., Horrocks I., *Enabling knowledge representation on the Web by extending RDF Schema,* http://www.ontoknowledge.org/, accessed on 2003-10-21.
[15] Klempe A., Warmer J., *Object Constraint Language: Precise Modeling with UML,* Addison-Wesley, 1999.
[16] Kruchten P. *The Rational Unified Process, An Introduction*, Addison-Wesley, 2000.
[17] Larman C., *Applying UML and Patterns, An introduction to Object-Oriented Analisys and design and The Unified Process 2nd edition,* Prentice Hall, 2001.
[18] National Comitee for Information Technology standards, Technical Comitee T2 (Information Interchange and Interpretation), http://logic.stanford.edu/kif/dpans.html, 1998, accessed on 2003-10-21.
[19] *Resource Description Framework (RDF) Model and Syntax Specification*, WWW Consortium, 1999, http://www.w3c.org/TR/1999/REC-rdf-syntax-19990222, accessed on 2003-10-21.
[20] Schwartz A., *The Semanic Web In Breadth*, http://logicerror.com/semanticWeb-long, accessed on 2003-10-21.
[21] *Annotated DAML+OIL (March 2001) Ontology Markup,* http://www.daml.org/2001/03/daml+oil-walkthru.html, accessed on 2002-03-01.
[22] *UML Specification version 1.4,* OMG 2001, www.omg.org, accessed on 2003-10-10.
[23] *XMI Specification version 1.1*, OMG 2000, www.omg.org, accessed on 2003-10-10.

Paper V

Extracting Initial UML Domain Models from DAML+OIL Encoded Ontologies

Extracting Initial UML Domain Models from DAML+OIL Encoded Ontologies

Ludwik Kuzniarz, Miroslaw Staron

Department of Software Engineering and Computer Science,
Blekinge Institute of Technology,
Box 520, Soft Center,
SE-372-25 Ronneby,
Sweden,
(Ludwik.Kuzniarz, Miroslaw.Staron)@bth.se

The paper presents and elaborates on an automatic method for creating an initial domain model using part of the knowledge contained in ontologies. It describes the method of how the initial domain model expressed in the Unified Modeling Language (UML) can be obtained in an automated way from ontologies encoded in DAML+OIL. The solution is presented in the context of the Unified Software Development Process, which uses UML as a modeling language. The elements necessary for construction of domain models are identified; a procedure for finding them in DAML+OIL encoded ontologies is described followed by suggestions for incorporation of the automatic domain model construction into a software development process.

1. Introduction

One of the main artefacts constructed during the analysis phase in the process of development of a software system is a domain model [17, 19]. Construction of the model requires expert knowledge about the domain and in many cases the system analyst responsible for the task has to put considerable time and effort in appropriate investigation of the domain and consulting domain experts. But the required knowledge may already exist expressed in the form of ontologies describing the domain under investigation and can be used to construct the initial domain model.

The paper explores the idea of automatic construction of an initial version of a domain model indicated in [30]. Further, it presents which knowledge from the existing ontologies that can be extracted for construction of the initial domain model and how the domain construction process can be automated.

The following assumptions concerning the representation of the information and processing technology were made: ontologies are expressed in DAML+OIL (DARPA Agent Markup Language and Ontology Interchange Language) [26], domain model is expressed in UML (Unified Modeling Language) [28], transformations are made using XSLT (Extensible Stylesheet Language for Transformations) [13]. They are discussed and justified in more details later - the main reasons for choosing UML and

DAML+OIL were their expressive power and the fact that they are accepted standards.

2. Basic notions

A software development process describes an approach to building software systems. It is usually based on a certain technology and describes activities to be performed and artefacts to be produced and is structured into development phases. In the paper the following characteristics of the process are assumed:

1. it includes a phase (usually called analysis) during which an abstract view or model of the domain is constructed,
2. it uses an object-oriented technology meaning that the model of the domain is expressed in terms of classes, their attributes and relationships, and
3. it is UML based that is all the artefacts produced are expressed in UML.

An example of such a process is the Unified Software Development Process [5], which is object-oriented and UML-based and the conceptual model is a basic artefact produced during the inception phase.

2.1. Domain model

Domain models contain concepts that exist in the real world and are important from the analyzed domain perspective to understand the problem, formulate the requirements and establish a basis for a software solution. The concepts are abstractions of real-world phenomena, which analysts and developers should take into consideration when analyzing the domain and later designing the software [21]. Domain models in UML-based software development processes are expressed as UML class diagrams. Abstractions for real world phenomena are called concepts and are expressed as classes. The concepts can have certain properties described as class attributes and relationships between classes.

Domain models are used to provide a common understanding for the analyzed problem domain, the part of reality that is significant for the system. The model allows identifying the important concepts and relationships between them, which exist in the real world. The detailed discussion of elements of domain models and their relevance in context of the acquisition process are discussed in [30].

2.2. Ontology

An ontology is a specification of conceptualization or an abstract model for some phenomena in the real world [11]. Ontologies are widely used in the mobile agent systems, where they provide a definition of knowledge which is used across heterogeneous agents to enable them interpret the same knowledge in the same way. Recently, ontologies are used to describe the resources in the Semantic Web [1, 2].

The usage in the Semantic Web resulted in a development of a constantly growing number of ontologies, which cover a number of domains.

Information contained in ontologies represents knowledge about a part of reality relevant to a certain domain. Even though there are ontologies that comprise knowledge across many domains, the majority contains knowledge specific for a certain domain. The ontologies are structured into two independent parts. The first, initial part contains information about the structure of knowledge and the second part contains the actual knowledge. The structure is defined by ontology classes describing concepts in the real world, properties of the classes, relationships between the classes and semantic information. The structure describes the instances of classes (objects) placed in the second part.

Classes can have class properties, which are properties that are expressed by another class and simple properties, which are the properties that can be expressed without referring to other classes. They are equivalent to concepts, attributes of the concepts and associations between concepts, which are used in domain models in software development process.

What is important, information contained in ontologies is already defined, widely used and machine processable. So deriving the domain model in an automated way can be desired and will provide a mean to ensure the consistency of the already used knowledge with the information that is used to build domain model. Moreover, the knowledge in ontologies is also used in the certain systems, which also means that it has been tailored to the specific domain.

The ontologies can be expressed in a spectrum of languages [6, 9, 12, and 14], however, the emerging standard is the DAML+OIL, which is a dialect of XML and is defined in [26]. The language is designed to be tool independent and able to express all elements which constitute an ontology while some of the other formats based on XML were found to be insufficient to express all information contained in ontologies [12, 15].

3. Domain model construction process

Typically domain models are constructed manually. The construction process involves such activities as finding candidate concepts and their later refinement (validation). The process demands from an analyst both expert knowledge about the domain under investigation (which may also require an extra time spent consulting domain experts) and a considerable time needed for constructing the model itself.

Several techniques aimed to help in domain model construction have been suggested such as finding noun phrase candidates, category lists checking or CRC (Class – Responsibilities – Collaborations) cards [4,5] all having a disadvantage of being time consuming as well as requiring expert knowledge and validation.

An alternative could be to search for a source on the required information which is already classified, structured and validated and can be automatically processed in order to obtain information required in the domain model. And it seems that such good candidates for the source of domain information are ontologies.

3.1. Domain model acquisition process

The process of constructing domain models as it is performed manually is depicted in Fig. 1 represented by a dashed line. The system analyst investigates the real world domain and produces the domain model. But to be able to perform this activity, the analyst must possess domain specific knowledge or to consult a domain expert [5]. In some cases the description of the domain does already exist in different form - ontology. It is produced by knowledge engineers, who defined an ontology describing this part of real world for other purposes. This knowledge can be used by software analyst to produce the desired domain model. To be able to use knowledge from ontologies, the software analyst must also be able to understand the structure of the knowledge in ontologies, and the elements used to define it. Thus the analyst must also be a knowledge engineer, which is usually not the case. Therefore assistance is required to discharge the system analyst from managing the ontology and the knowledge contained in it - what is depicted in Fig. 1.

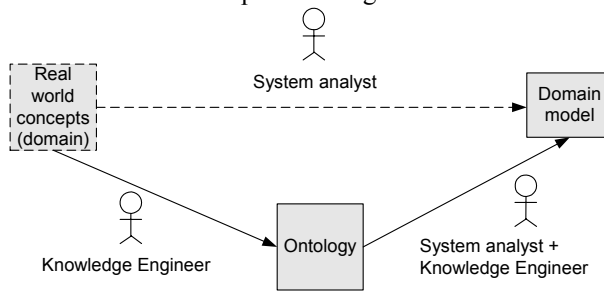

Fig. 1. Manual domain model construction from Ontology

The assistance can be performed by an automatic model acquisition procedure. Its placement in the context of domain model production from ontologies is presented in Fig 2. It is depicted as a rectangular actor at the bottom of the figure.

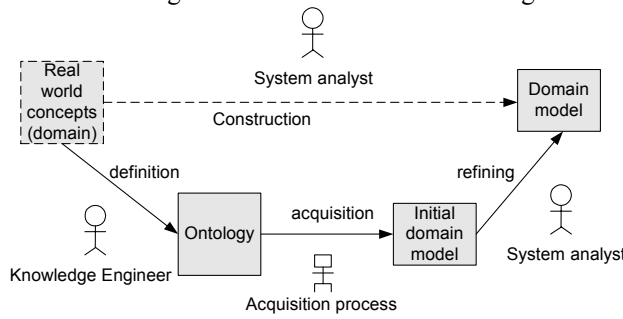

Fig. 2. Automatic initial domain model construction in the context of domain modeling

The advantage of the acquisition process is that it releases the system analysts from understanding formats used for definition of ontologies and at the same time providing a way of creating an initial version of the domain model, which is consistent with the knowledge analyzed by the knowledge engineers. This initial version of domain model must be refined by the analyst to tailor it to the specific

requirements for the developed software system. Therefore the presented schema allows performing the activity showed as dashed arrow in a more formal and easier way, which introduces consistency and reuses the existing knowledge and at the same time not demanding new skills from analysts.

3.2. Characteristics of the acquisition process

An important characteristic of a good acquisition process is that it should not be bounded to any specific tool used for domain modeling or ontology engineering. The process should be tool independent. The independence should be supported by using technologies that are open and unbounded, but at the same time supported by more than one tool.

Furthermore, the acquisition process should be based on technologies, which are not restricted by proprietary laws. An industrial standard for a language that fulfills this requirement is Extensible Markup Language (XML) and a set of technologies built on top of the language. For UML, there is a dedicated dialect developed for internal representation of UML models. The dialect (Extensible Metadata Interchange - XMI) [29] is maintained by OMG, and therefore is not bound to any specific UML tool vendor. It is supported by a lot of tools on the market (i.e. Rational Rose, ArgoUML, etc.). The XMI dialect reflects the specification of an abstract syntax of the Unified Modeling Language - its metamodel. Every model can be represented by a set of tags, which are structured according to the metamodel. The XMI specification also provides unified way of producing XMI documents from the existing models as well as creating models from their XMI representation.

For ontologies, the language based on XML is DAML+OIL (described in more detail in section 4). It is also a recognized standard for defining ontologies. It provides a necessary set of tags to encode the ontologies. It is also designed to be interpreted by autonomous mobile agents as well as tools for the Semantic Web. DAML+OIL provides a mechanism to encode both the structure of the knowledge and the actual knowledge in a unified way in the same documents.

Both XMI and DAML+OIL are pluggable into the existing software developed for UML and ontologies development respectively. Therefore the standards are widely accepted and implemented. Taking advantage of them ensures tool consistency in terms of interpretability and acceptance of the standards.

4. Realization of the construction process

The realization of the construction process is divided into two stages, as depicted in Fig. 3. The first stage is an automated acquisition of knowledge, and the second stage is an automatic translation of the acquired knowledge from ontology description language into internal representation of UML models. The role of the acquisition is to extract the elements of the ontology, which are of interest during domain analysis. In particular it leaves the structural information contained in the original ontology such as classes, class relationships and properties. This information is still expressed in the form of DAML+OIL encoded ontology which is then translated into the initial

domain model. The role of the translation process is to change the representation of the acquired knowledge from ontology definition language to the UML representation language. It is defined by a mapping between the corresponding concepts in domain models and ontologies. The semantic information is intentionally not included. Including it in domain modeling requires some extensions of the language used for domain modeling to express it. The extensions should involve such elements as stereotyped dependencies, unnamed collection classes, etc. [3, 7, 8].

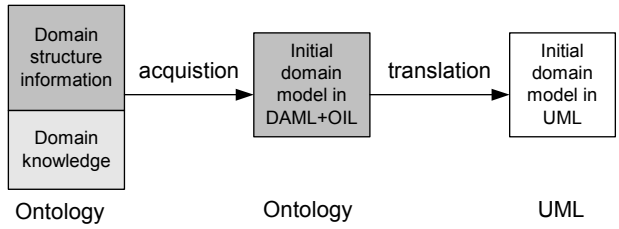

Fig. 3. Schema of construction of initial domain model

Given the two representation languages - XMI and DAML+OIL, the process can be performed by Extensible Stylesheet Language for Transformations (XSLT) [13] which is a technology dedicated for transformations of XML-based languages. The XSLT technology is primarily designed for transforming the structure of XML document [13]. It is a high level declarative language, which describes rules how to transform the XML documents. The details of the implementation of how these XSLT transformations work are hidden behind the XSLT processors, which are responsible and designed for interpreting and realizing XSLT stylesheets (definitions of XSLT transformation rules). With the use of this technology both acquisition, which is a transformation from DAML+OIL to DAML+OIL and translation, which is a transformation from DAML+OIL to XMI are performed both described by XSLT stylesheets as depicted in Fig. 4.

The dark gray rectangles depict ontologies. The left-hand ontology is the original one, and the ontology in the middle, which is the result of the acquisition process and contains only information about domain structure (domain model, but defined as DAML+OIL encoded ontology and so are gray shaded). The light gray color designates elements that belong to the XSLT technology and which take part in transformations. Stylesheets are constant for all ontologies and are bound to the process of domain model construction. The details and the specification of the stylesheets can be found in [18]. The resulting domain model (white rectangle) contains the information about the structure of the domain (encoded in XMI), which can be imported to the UML tool used. The advantage of the proposed acquisition process is that to execute it no prior knowledge is needed about the XSLT technologies it is only necessary to run a standard XSLT application with the supplied stylesheets and DAML+OIL encoded ontology. What is more the stylesheets are written once and there is no need to alter them unless the conversion rules ere changed.

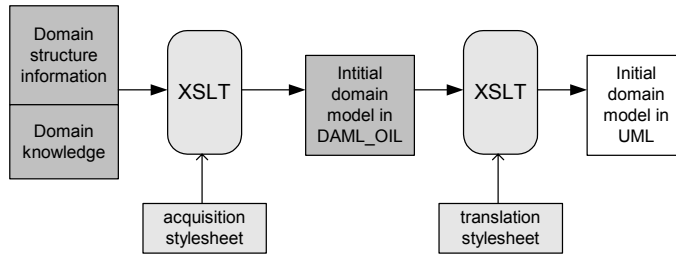

Fig. 4. XSLT transformations for domain model construction

The representation of information in DAML+OIL and XMI differs significantly, and thus the translation is much more complicated than in the case of the acquisition. XMI reflects the structure of the metamodel and therefore the translation not only maps corresponding concepts from DAML+OIL to the equivalents in XMI, but also adds some properties to the resulting elements. The properties are introduced with regard to the semantics of DAML+OIL elements and according to the XMI specification. The detailed description of mappings and introduced properties for elements previously described can be found in [18].

The acquisition is not only the process of removing unnecessary information but it makes necessary adjustments to the original ontology. The adjustments do not change the semantics of the ontology, but make the translation to XMI easier. The changes are aimed to group elements concerning the same concept, or change equivalent elements (like restrictions on "Cardinality", which are changed to restrictions on both "minCardinality" and "maxCardinality"). Grouping elements concerning the same concept allows translating the ontology in a more structured manner, and makes the realization of the translation straightforward.

5. Including the construction process into a software development process

The right place in software development process, where the acquisition process should be introduced is the initial construction of the domain model. This activity is performed at the beginning of transition from requirements to use cases. It usually takes place (depending on an individual variation of USDP) during the inception or early elaboration phase. Since prior to the knowledge acquisition, the search for the required ontology need to be performed, this activity should be performed during inception phase, in parallel to the activity of finalizing requirements capture. Such a placement of ontology search has two main advantages. The first one is that at the end of requirements gathering activity, the analysts know what is the domain of system, so the proper ontology can be found. The second advantage is that if the search results in multiple ontologies, they can be evaluated according to the requirements for the built system. A helpful resource to search for ontologies is [27], where a rich set of ontologies is catalogued.

Resources, in sense of time, need to be devoted for seeking the proper ontology, but the resources are compensated by time that is saved for analysis of the domain and construction of the initial domain model. Nevertheless, the manual refinement of the initial version of this version of conceptual model cannot be eliminated since as every software system deals with different matters so not all elements should be kept in the domain model.

Some concepts identified in the initial domain model should be included only as terms in the glossary document, and some should be removed from the documentation at all. What is more, some concepts are missing in the ontology, and they must also be added manually. Furthermore, for software which is meant to work on a verge of two domains, the manual merge of two domain models may be required.

A proper construction of the initial domain model is important for the subsequent activities, which are based on domain model and result in artefacts, which again are crucial for some activities. However, domain modeling is crucial, because it forms a "vocabulary" of the system, which is used throughout the development process [19, 21]. The automated acquisition process helps to produce domain models, which are consistent with the existing knowledge about the domain, and what is more, the knowledge which is used, so it has been informally proved by others to be useful and properly defined.

6. Example

The practical usage of the domain knowledge acquisition process can be presented on the example of a small banking system. The example shows how the usage of this process can result in generating an initial version of a domain model, which can later be adjusted according to the specific software. The problem solved in this example is the banking system which allows managing customer accounts and providing a possibility to withdraw and deposit money on the accounts.

One of the first activities performed during the development of this software is to analyze the domain of a bank. However, instead of using one of the manual techniques presented in section 2, we try to use the ontology, which has been designed for use with mobile agents. For simplicity of the example, the knowledge that is contained in the ontology is not shown, and the ontology is abridged and visualized.

The ontology consists of two parts. The first one (at the top of Fig. 5) defines the knowledge about domain structure. It contains concepts (classes) and relationships between concepts. The second part (the lower part of the figure) contains the knowledge about objects in this domain. In case of this example, the knowledge is an instance of class Customer, and its connection to the instances of classes Account and Loan. The domain describes simple relationships between accounts, loans and customers of the bank. The properties are omitted for the sake of simplicity and because the discussion on the way to include them is summarized in [6].

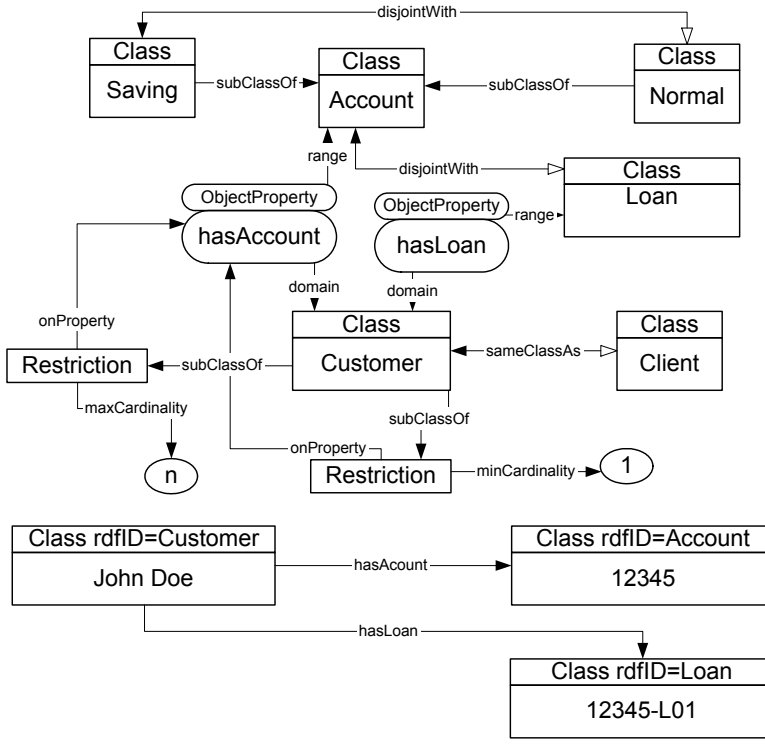

Fig. 5. Banking ontology

From the original ontology, the acquisition process produced the ontology presented in Fig. 6. Concepts that have been omitted since they are not parts of domain models are:

- existing objects within the domain:
  - instance of a customer (John Doe),
  - instance of an account (12345),
  - instance of a loan (12345-L01)
- semantic information (relationship sameClassAs between classes Client and Customer, relationship disjointWith between classes Saving and Normal)

The filtration also omits other information, but for the sake of simplicity of this example, it is not presented in the paper, but included in the report [18]. The translation process, which has been performed on the acquired ontology, produced an initial version of the conceptual model for the modeled banking system.

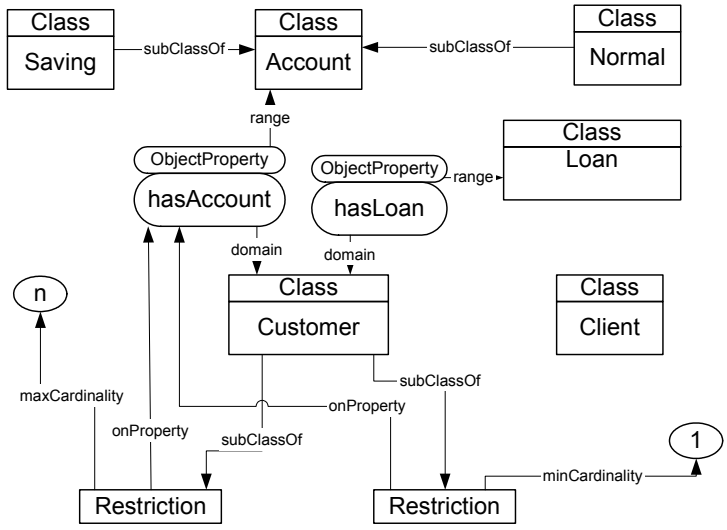

Fig. 6. Filtered banking ontology

The domain model as UML class diagram is presented in Fig. 7. This domain model can later be changed by the analyst, who refines the initial version produced so far. The refinement can involve adding some concepts, which were not included in the ontology, but are crucial for the banking system which we build. What is more, the ontology contained no such concept as our modeled system.

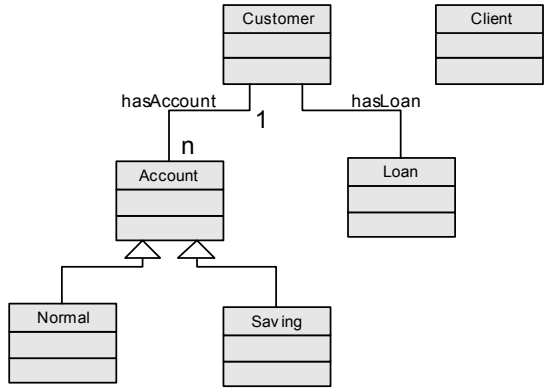

Fig. 7. Initial domain model for banking ontology

Therefore, this concept must also be added. The resulting domain model (after multiple alterations) can be used in the later stages of software development. It is presented in Fig. 8.

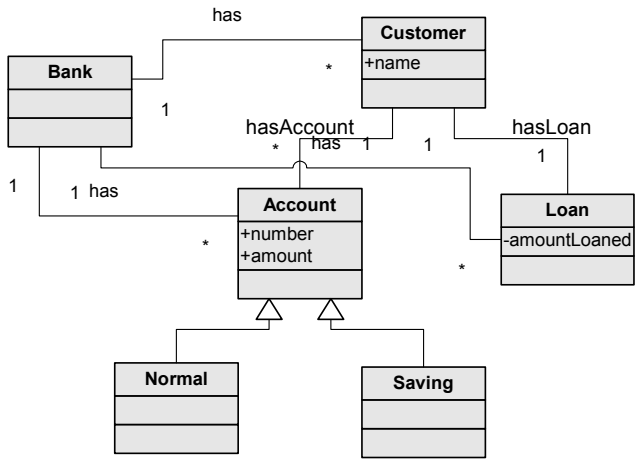

Fig. 8. Refined domain model for banking system

The analyst removed concepts that were redundant (class Client) and introduced some attributes that were not shown in the ontology.

7. Related work

Attempts to integrate ontology engineering and UML have already been performed. However, the available solutions are not suitable for the purpose of the domain knowledge acquisition process. Baclawski et al. [3] present an extension to UML supporting ontology engineering in UML. The extension is based on built-in UML extension mechanisms as well as changes in the UML metamodel (the UML specification). The extension was developed to provide the necessary constructs required for modeling ontologies in the customized version of UML[1]. However, since domain models require only certain elements from ontologies, the extension based on changes in the UML metamodel is too demanding in terms of knowledge and experience in ontologies that analysts constructing domain models should have. Furthermore, using the extended UML to express domain models may be in contradiction with the purpose of the models, which are to be used as a basis for understanding the domain of the constructed system (not the ontology describing the domain). The domain knowledge acquisition process omits this problem.

Another approach to integration of UML and ontologies was presented by Cranefield and Purvis [8] who propose a method of using the standard UML constructs for modeling ontologies. Their method is based on similar technologies and assumptions as in the domain knowledge acquisition process presented in this paper, but it can be seen as the reverse process to it. The method of Cranefield and Purvis uses standard UML class and collaboration diagrams to model ontologies, which then

[1] i.e. all elements of ontologies could be expressed in the customized UML

can be expressed in XMI which is used to generate ontologies expressed in RDF (Resource Description Framework – a predecessor of DAML+OIL, [22-24]). The RDF encoded ontologies can be used by Java programs, which contain parts generated from the XMI documents which contain the ontologies. The generation of ontologies and parts of Java programs from XMI documents is based on the XSLT technology as in the domain knowledge acquisition process.

8. Conclusions and further work

In the paper a process of automatic construction of initial versions of domain models is presented. It is based on the acquisition of the knowledge already existing in DAML+OIL encoded ontologies. What can be acquired is discussed, followed by a method of how the acquired information can be included into a UML based development process. The acquisition process is based on XML technology: firstly both DAML+OIL itself and XMI (representation standard for UML model) are XML dialects and secondly the transformation is encoded as XSL stylesheet containing translation rules for XML encoded data. The result of the transformation is an XMI encoded class diagram so it can be directly imported to UML tools.

The advantage of automatic construction of the initial domain compared to manual construction is that the obtained model is consistent with the expert domain knowledge expressed in the ontology guaranteed by the construction algorithm. However at the present experimental state the transformation filters some knowledge such as the full notion of collections and restrictions and the obtained domain model does not express all the knowledge about the domain. Partially this is a desired feature because including all features expressible in DAML+OIL would require extending the UML metamodel [3] or using the Object Constraint Language (OCL) and some of them were just omitted for technical reasons and are the subject of current research on the transformation tool. The current research is also focused on incorporation of the transformation into the UML tool so that it would be possible to import DAML+OIL encoded ontologies to obtain a class diagram expressing a domain model. A further research into the problem of refinement of the initial domain model based on the domain of the system is performed.

9. References

1. Barners-Lee T., Hendler J., Lasilla O., The Semantic Web, Scientific American, May 2001,
2. Barners-Lee T., Semantic Web Road map, WWW Consortium September 1998, http://www.w3c.org/DesignIssues/Semantic.html,
3. Baclawski K., Kokar M. K., Kogut P.A., Hart L., Smith J., Holmes III W. S., Letkowski J., Aronson M., Extending UML to Support Ontology Engineering for the Semantic Web, In the Proceedings of The Fourth International Conference on Unified Modeling Language, Toronto 2001, pp. 342 ff,

4. Beck K., Cunningham W., A laboratory of teaching object-oriented thinking, SIGPLAN Notices vol. 24 (10/1989), pp. 1-6,
5. Booch G., Unified Software Development Process, Addison-Wesley, 1998,
6. Chang W. W., A Discussion of the Relationship Between RDF-Schema and UML, W3C note, document http://www.w3.org/TR/1998/NOTE-rdf-uml-19980804,
7. Cranefield S., UML and the Semantic Web, In the Proceedings The International Semantic Web Working Symposium, SWWS 2001 Symposium, Stanford 2001, pp. 113-130,
8. Cranefield S., Purvis M., UML as an Ontology modeling language, in Proceedings of The Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99), 1999,
9. Fenzel D., The Semantic Web and its languages, IEEE Intelligent Systems, November/December 2000,
10. Fenzel D., Horrocks I., Van Harmelen F., Decker S., Erdmann M., Klein M., OIL in a nutshell, Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2000), volume 1937 of Lecture Notes in Artificial Intelligence, Springer Verlag 2000, pp. 1-16,
11. Gruber T. R., A Translation approach to portable ontology specification, Knowledge Acquisition 5(2), October 2000, pp. 199-220,
12. Hjelm J., Creating Semantic Web with RDF, Wiley 2001,
13. Kay M., XSLT, Programmer's reference, Wrox Press, 2000,
14. Klein M., Fenzel D., Van Harmelen F., Horrocks I., The Relation between Ontologies and Schema-languages: Translating OIL-specifications in XML-Schema, In the Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence ECAI-00, Berlin, Germany August 20-25, 2000, http://delicias.dia.fi.upm.es/WORKSHOP/ECAI00/7.pdf,
15. Klein M., Broekstra J., Decker S., Fenzel D., Harmelen F., Horrocks I., Enabling knowledge representation on the Web by extending RDF Schema, http://www.ontoknowledge.org/
16. Klempe A., Warmer J., Object Constraint Language: Precise modeling with UML, Addison-Wesley, 1999,
17. Kruchten P. The Rational Unified Process, An Introduction, Addison-Wesley, 2000,
18. Kuzniarz L., Staron M., Hellman E. Extracting information about domain structure from DAML+OIL encoded Ontologies into UML Domain Models, Blekinge Institute of Technology, Research Report 2002:02, ISSN 1103-1581, 2002,
19. Larman C., Applying UML and Patterns, An introduction to Object-Oriented Analysis and Design and The Unified Process 2nd edition, Prentice Hall, 2001,
20. Rational Corporation website, www.rational.com,
21. Rational Unified Process User Guide, Rational Corp, www.rational.com,
22. Resource Description Framework (RDF) Schema Specification 1.0, WWW Consortium, 2000, document http://www.w3c.org/TR/2000/CR-rdf-schema-20000327,
23. RDF Model Theory, W3C Working Draft, WWW Consortium, September 2001, http://www.w3c.org/TR/2001/WD-rdf-mt-20010925,

24. Resource Description Framework (RDF) Model and Syntax Specification, WWW Consortium, 1999, http://www.w3c.org/TR/1999/REC-rdf-syntax-19990222,
25. Schwartz A., The Semantic Web In Breadth, http://logicerror.com semanticWeb-long,
26. Annotated DAML+OIL (March 2001) Ontology Markup, http://www.daml.org/2001/03/daml+oil-walkthru.html,
27. DARPA Agent Markup Language Web site, http://www.daml.org
28. UML Specification version 1.4, OMG 2001, www.omg.org,
29. XMI Specification version 1.1, OMG 2000, www.omg.org,
30. Kuzniarz L., Staron M., Generating Domain Models from Ontologies, In the proceedings of OOIS 2002, Springer-Verlag LNCS vol. 2425, 2002, pp. 160-166.