Transformational Stereotypes: A Support for Transforming UML models

Miroslaw Staron[1)], Ludwik Kuzniarz[2)]

[1)] Department of Applied IT
IT University in Göteborg
miroslaw.staron@ituniv.se

[2)] Department of Systems and Software Engineering, School of Engineering
Blekinge Institute of Technology
ludwik.kuzniarz@bth.se

Automatic model transformations can be regarded as the main element of model driven software development. For models created with the use of the Unified Modeling Language (UML) the task of defining transformations is believed to have been accompanied with creation of auxiliary elements that allow tackling the problems of ambiguity of UML. One of such auxiliary elements can be a stereotype. Stereotypes created for the purpose of supporting transformations have specific properties and should be used in a specific way. In this paper we elaborate the specific type of stereotypes – *transformational stereotypes*. Based on the observations and an investigation of several UML profiles that are used in industry, in this paper we show properties of such stereotypes and provide guidelines on how the transformational stereotypes should be used and created.

1 Motivation

Effective usage of models in course of software development requires procedures for their automated processing. These procedures are usually referred to as *model transformations*. One of the most common visual notation for expressing models in a graphical way is the Unified Modeling Language (UML, [1]), currently at its second revision. UML was created in order to provide a uniform notation for modeling all kinds of software artefacts – from very abstract conceptual models to very detailed design blueprints. This generality of UML raises problems in the context of automatic model transformations since the transformations require presence of additional information in models to perform their tasks correctly. The additional information is required since standard UML model elements do not have properties which are specific for the application domain. Therefore effective automation of model transformations requires making UML models more specific and thus more precise. This in turn can be done by using stereotypes (since they are an extension mechanism inherent in UML) as a means of adding necessary information to existing model elements.

In the course of our prior industrial case studies we have observed that code generation stereotypes can be perceived as a special case of another kind of stereotypes – transformational stereotypes. *Transformational stereotypes* can be defined as stereotypes used in conjunction with model transformations to provide additional details for model elements being transformed. Transformational stereotypes support model transformations. Due to their purpose these stereotypes have certain common properties and usage scenarios. In this paper we introduce the transformational stereotypes, describe their intent and provide usage guidelines for their usage and creation.

Our work was motivated by an investigation of several UML stereotypes used in practice showed that 24% of stereotypes were used for improving code generation purposes (c.f. [2]). Since the code generation stereotypes are a sub-category of the transformational stereotypes the usage of transformational stereotypes could be even wider thus their proper creation needs to be effective. In the studied industrial application of MDA we found that the transformations require more than 20 times more effort for creation than stereotypes and that the stereotypes play a primary role in the development of transformations of UML model (c.f. [3]) which requires a correct and efficient usage of stereotypes dedicated for that purpose.

While specifying transformations, in the course of discussions we follow a simplified version of defining transformations presented in [4, 5]. In particular we focus only on the Object Constraint Language (OCL) in the source and target conditions in the course of our reasoning. In order to avoid confusion with the full transformation specification we refer to these as only pre- and post-conditions of transformations without the loss of generality of the discussions in the paper.

The outline of the paper is as follows. First we outline the most relevant related work in the fields of model driven software development, model transformations and the use of stereotypes in software development in Sect. 2. Second we overview the methods used in our study which resulted in identifying and descrining properties of transformational stereotypes in Sect. 3. Transformational stereotypes are presented in Sect. 4. The properties of transformational stereotypes are given in Sect. 5. The properties are followed by usage guidelines elaborated based on these properties and the observations of current usages of the transformational stereotypes in Sect. 6. The conclusions and further work is presented in Sect. 7.

2 Related work

Stereotypes have been given a special attention together with the idea of the Model Driven Architecture (MDA, [6]) which is gaining popularity. The idea of models being main assets in modern software development strives for more precise models and more flexible languages to create them. As stereotypes are the main extension mechanism providing some flexibility for UML, they were evaluated in several ways by analyzing different ways of using and defining them (e.g. [7-10]). Although the stereotypes are found to be very suitable for lightweight language customization, none of the analysis performs a formal empirical study on stereotypes, which is a presented in our research.

The authors of [11] use stereotypes as compartments for storing information which was specific to the domain (information systems) for which the models where created. Their stereotypes did not contain constraints, but information which was used while transforming platform independent models into platform dependent models. The authors, however, did not explicitly identify a new type of stereotypes which is done in this paper.

A particular importance of transformations and stereotypes related to them was presented in [3, 12]. The set of stereotypes used in the studied organization was intended to increase the amount of code generated from models to 100% for a specific domain. As these stereotypes were dedicated for supporting the transformations, they were incorporated into the study presented in this paper.

Transformational stereotypes discussed in this paper can be opposed to the traditional use of stereotypes presented in [13] where the stereotypes are used for providing means of additional classification of objects according to their role in designs. The stereotypes considered in this paper are not intended to play that role since they only implicitly provide classifications discussed by Wirfs-Brock. Stereotypes used as a secondary classification mechanisms should be considered in the context of modelers and not their tools (e.g. [14, 15]).

The UML specification [16] itself proposes the basic way of documenting stereotypes. It was used as a basis for elaboration of the templates for documenting transformational stereotypes presented in this paper.

3 Method

The stereotypes were identified during a previous study on an industrial MDA realization [12]. The idea of transformational stereotypes was further investigated as an additional analysis of results of experiments with classifications of stereotypes [2] whereby we elaborated properties of stereotypes in relation to their purposes. The process is presented in Fig. 1.

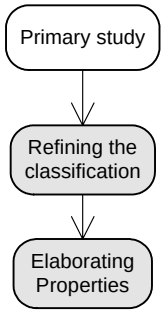


Fig. 1. Process of identifying the transformational stereotypes and elaborating their properties

The focus of this paper is on the activities with gray background. The process highlights the fact that the findings presented in this paper are an additional analysis (per-

formed a posteriori) of the primary study reported in [2]. The primary study was performed on the sample of 98 stereotypes with an auxiliary study validating the instrumentation and study method. The additional analysis resulted in improved grouping of stereotypes. The new grouping was used as an input for analysis of stereotypes hence allowing for an identification of a set of common properties of transformational stereotypes.

In order to elaborate properties of stereotypes, the stereotypes had to be grouped according to some criteria. The criteria which were used in our study were derived from the classification of stereotypes according to their expressiveness by Berner et al in [17]. The expressiveness is the amount of changes in syntax and semantics a stereotype introduces to the base model element. In their work the authors distinguished between four categories of stereotypes:

1. *Decorative stereotypes*, i.e. stereotypes which do not change the semantics of a language element, but change its concrete 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. The classification aims in answering the question – "What changes does the stereotype make to the base model element?"

The classification provided also a starting point for elaborating properties of transformational stereotypes; depending on the category in the classification the transformational stereotypes can have different properties.

4 Transformational stereotypes

The identification of the transformational stereotypes as a specific category of stereotypes allows creating a taxonomy of stereotypes which increases the level of details of the classification of stereotypes presented in [18]. The taxonomy is presented in Fig. 2.

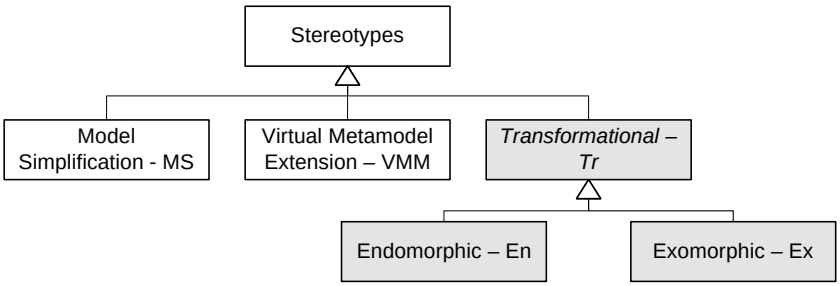


Fig. 2. Taxonomy of stereotypes introduced in [18] and refined in this paper.

The taxonomy presents the new category of stereotypes with its subcategories (grayed in Fig. 2.). Subcategories of transformational stereotypes are derived from the taxonomy of model transformations presented in [19]. In particular the authors distinguish between:

- *endomorphic* transformations intended to transform one UML model into another – e.g. refactoring or database structure creation supported by UML profiles ([20]);
- *exomorphic* transformations intended to create non-UML artefacts from UML models – e.g. code generation;

Both kinds of transformations can be supported by the transformational stereotypes and both subcategories of stereotypes have common properties (associated with the supercategory of transformational stereotypes) and disjoint properties (associated with each of the subcategories).

Although there exists third kind of model transformations – creational transformations – stereotypes which are used in these transformations are stereotypes which can be categorized as other kind of stereotypes since they are not directly supporting the transformations, but they are a result of the transformations.

A specific case of exomorphic transformational stereotypes are stereotypes used for improving code generation – *code generation stereotypes* – presented in [18]. These stereotypes are used for transformations that result in source code for a specific programming language. Due to the fact that this purpose (code generation) was considered in our previous studies we explicitly mention the code generation stereotypes. These stereotypes already were identified to possess common properties and these properties were the initial set of properties of the transformational stereotypes complemented with new properties as the transformational stereotypes are a supercategory of the code generation stereotypes. In this paper we refer to code generation stereotypes as exomorphic stereotypes to explicitly emphasize the fact that code generation is not the only possible use of these stereotypes – another use is the generation of Ontologies from UML [21].

4.1 Notion of transformational stereotypes

From the perspective of UML the transformational stereotypes are structured in the same way as other stereotypes – they can be created using the same elements as all

stereotypes. The differences making it possible to distinguish these transformational stereotypes from the rest of stereotypes are visible at the usage level characterized in this section.

There are two main functions of the transformational stereotypes – make it possible to:

- add details to stereotyped model elements (these details are necessary for automating model transformations)
- simplify the code for checking the pre- and post-conditions of transformations

The two functions of transformational stereotypes are related to the way in which model transformations are specified and in which the transformations are conducted.

Adding details. This function is fulfilled by tag definitions created as part of the stereotypes. Automatic transformations of UML models need more details than the standard UML modeling elements usually have. The standard modeling elements are for general use while the transformations are intended for specific usages. In other words the standard constructs are used in different contexts (e.g. classes are used to depicts concepts in domain modeling, software classes or database entities) while model transformations transform specific usages of constructs into another specific usages constructs (e.g. a transformation of ER entities into database tables – both modeled with UML classes).

The details added to the modeling constructs are used by model transformations for getting information as input for the transformation. An example of using the tag definition for the first situation is the transformation of services into components with help of stereotype «Service» presented in Fig. 3:

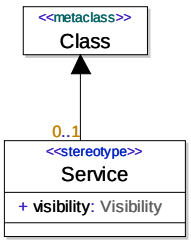


Fig. 3. Stereotype «Service» - example of adding details with a tag definition

The transformation, based on the value of the tag visibility can transform the service into a business component or into a service component. The information contained in the tag definition allows automating the transformation process; otherwise the modeler would have to mark the services which are intended to be business components, run the transformation, mark the components which are intended for service components, and run another transformation (which would still make the process half-manual).

Simplifying transformation code. The code of the transformation needs to take care of the entry conditions of the transformation. The entry conditions can be specified as pre-conditions written as OCL constraints. The pre-conditions, however, can get very

complex if the input model for the transformation is very restricted. Therefore, these pre-conditions can be simplified by specifying the well-formedness rules for stereotyped model elements and then using the presence of the stereotype as a pre-condition of the transformation. This allows breaking down the complexity of the transformations which is one of the foundations for industrial adoption of model transformations.

For example, let us consider a trivial transformation which transforms one class into another (a business component into an EJB component). The code for the pre- and post-conditions for the transformation can be:

```
context BCtoEJB(x: Class): Class
 Pre: x.allDependencies()->
       forAll(d|d.supplier.oclIsKindOf(Class)
 Post: result.allDependencies()->size() = 0
```

The pre- and post-conditions state that the class to be transformed can only be dependent on other classes and that the transformed class is not dependent on other classes. The same can be expressed in another way, which is more readable for the modeler with the help of stereotypes «BC» and «EJB» for business components and Enterprise Java Beans respectively. The pre- and post-conditions for the transformations can look like[1]:

```
context BCtoEJB(x: Class): Class
 Pre: x.isStereotyped(BC)
 Post: result.isStereotyped(EJB)
```

And the stereotypes «BC» and «EJB» can have the following constraints defined:

```
context BC:
 Inv: baseClass.allDependencies()->
       forAll( d | d.supplier.oclIsKindOf(Class)

context EJB:
 Inv: baseClass.allDependencies()->size() = 0
```

Using the stereotype lets also improve the effectiveness of modeling since the violation of the above constraints would be reported as errors during the modeling and not during the transformation (as in the previous case). This would better reflect the reality as the constraints are related to the well-formedness of models rather than the transformation – the transformation transforms classes that are explicitly used as business components and not all classes that are dependent only on other classes. In this way using the stereotype also allows checking whether the transformation was invoked in an appropriate situation. Such issues were identified as being important from the user perspective while introducing MDA into industrial software development in [12].

[1] We assume the existence of the OCL operations *isStereotyped(…)* and *allDependencies(…)*

5 Properties of transformational stereotypes

The two roles of transformational stereotypes – adding details and simplifying transformation code – are coupled to two modeling constructs associated with the notion of stereotypes – tag definitions (attributes of stereotypes) and constraints. Therefore the starting point in elaborating the properties was the analysis of the properties associated with the categories in the classification of stereotypes according to their expressiveness. The properties are discussed in relation to the particular subcategory of transformational stereotypes.

5.1 Exomorphic stereotypes (Ex)

There are two categories of code generation stereotypes which were identified in our study (c.f. [2] for details) – restrictive and redefining stereotypes. A close investigation of these two types shows that all exomorphic stereotypes possess common properties:

- base model elements (i.e. the elements the stereotype is extending) are usually concrete meta-classes from the UML metamodel as the concrete meta-classes have rules on how they should be used in the designs (some also have some standard code generation rules built into modeling tools),
- "templates" for code generation as part of the semantics of the stereotype – sometimes the "template" can be defined as a specific tagged value (provided that the code generator can interpret it), and
- no additional graphical icons defined for their presentation.

A way in which the templates for code generation are specified depends on the used code generator. The mechanisms of code generation (or exomomorphic transformations in general) should be investigated before creating exomorphic stereotypes (except, naturally, for profiles intended to be standardized – e.g. the UML Profile for CORBA – in which the templates can be given only in textual form as the profiles are not dedicated for any specific tool – e.g. specific code generation templates can be found in [22, 23]).

Redefining stereotypes. The stereotypes in this group redefine the semantics of their base elements. The redefinition of semantics makes the stereotyped instances of base model elements completely different from non-stereotyped ones. The semantics of the stereotypes of this type is defined by the semantics of the element in the target language which the stereotype represents. The redefinition of the semantics usually requires that the stereotyped elements are used in models only with other stereotyped elements. The redefining code generation stereotypes have:

- tag definitions which have custom-defined data types,
- constraints restricting the usage of the stereotyped elements so that they can be used only with other stereotyped elements,
- semantics of the stereotyped elements that differs significantly from the semantics of the semantics of the model element being extended.

The data types of tag definitions are defined as part of the same profile. There are, however, certain exceptions as not all tag definitions have custom defined data type. A few stereotypes in the study had tag definitions which had standard data types – in most cases Boolean.

An example stereotype of this type is «CORBATypedef» from the UML profile for CORBA. It is a stereotype that can be applied to classes. The stereotyped classes can be used only with relation to classes that are stereotyped with other stereotypes from the CORBA profile. Furthermore, the stereotyped classes cannot have attributes and the code to be generated for the «CORBATypedef» stereotyped classes is very different than the code to be generated from non-stereotyped classes.

Restrictive stereotypes. This type of stereotypes is used if an element in the target programming language is similar to an existing model element in UML although it lacks certain properties. Using model elements stereotyped with stereotypes of this type is allowed in most of the cases when the base model element can be used, but there are restrictions on the usage of the stereotyped elements – c.f. [17]. Stereotypes of this type have:

- tag definitions which types are defined by the creator of the stereotype or data types defined as part of the UML standard,
- constraints "restricting" the usage of the stereotyped elements in certain situations in which the base modeling element can be used,
- semantics making the semantics of the extended model element more precise.

The data types used for tag definitions can be both custom-defined (most often) or built-in.

As example stereotype of this type is «CORBAStruct» from the UML Profile for CORBA. The stereotype is applied to classes in UML and makes the semantics of classes more precise – e.g. that all the attributes should be public. Its constraints state that the «CORBAStruct» stereotyped class cannot be used in some situations when the non-stereotyped class can be used.

5.2 Endomorphic stereotypes (En)

There are three categories of stereotypes according to Berner's classification that were identified among endomorphic stereotypes – descriptive, restrictive, and redefining. A close investigation of these three types shows that all endomorphic stereotypes possess common properties:

- base model elements (i.e. the elements the stereotype is extending) are concrete meta-classes from the UML metamodel (similar to exomorphic stereotypes which sometimes extend also abstract meta-classes), and
- description of the mapping – i.e. the description of which elements are created based on compound elements of the stereotype model element – e.g. which elements are created based on attributes of the transformed class.

Descriptive stereotypes. These stereotypes are used to add new properties to model elements which are used by model transformations. The properties of endomorphic descriptive stereotypes are:

- types of tag definitions being both standard data types specified in the UML metamodel and custom defined data types defined by the creator of the stereotype,
- base classes are concrete meta-classes from the UML metamodel,
- no constraints restricting the usage of the stereotyped elements, and
- usually no icons.

An example of this kind of stereotype is the stereotype «Service» presented in Sect. 4.2 – Fig. 3. The stereotype contains one tag definition and its type is custom-defined – Visibility. Its primary role is to provide more information for a model transformation while there are no restrictions imposed on stereotyped model element by constraints specified as part of the stereotype.

Restrictive stereotypes. This type of stereotypes is used if an element in the target programming language is similar to an existing model element in UML although it lacks certain properties. Using model elements stereotyped with stereotypes of this type is allowed in most of the cases when the base model element can be used, but there are restrictions on the usage of the stereotyped elements – c.f. [17]. Stereotypes of this type have:

- tag definitions which types are defined by the creator of the stereotype or data types defined as part of the UML standard,
- constraints "restricting" the usage of the stereotyped elements in certain situations in which the base modeling element can be used,
- semantics making the semantics of the extended model element more precise.

The data types used for tag definitions can be both custom-defined (most often) or built-in.

As example stereotype of this type is the stereotype «BC» referred to in Sect. 4.2. The constraints specified for the stereotype is restricting the usage of the stereotyped model element – it can be dependent only on classes and no other elements.

Redefining stereotypes. The stereotypes in this group redefine the semantics of their base elements. The redefinition of semantics makes the stereotyped instances of base model elements completely different from non-stereotyped ones. The semantics of the stereotypes of this type is defined by the semantics of the element in the target language which the stereotype represents. The redefinition of the semantics usually requires that the stereotyped elements are used in models only with other stereotyped elements. The redefining code generation stereotypes have:

- tag definitions which have custom-defined data types,
- constraints restricting the usage of the stereotyped elements so that they can be used only with other stereotyped elements,
- semantics of the stereotyped elements that differs significantly from the semantics of the semantics of the model element being extended.

The data types of tag definitions are defined as part of the same profile. There are, however, certain exceptions as not all tag definitions have custom defined data type. A few stereotypes in the study had tag definitions which had standard data types – in most cases Boolean.

Example stereotypes of this type are stereotypes which are used in a profile for database development presented in [20] or the domain specific profile studied in [12]. An example stereotype from the previous is the stereotype «ERD_Entity» which redefines the semantics of its base model element – class – so that it can only be used with certain stereotyped elements (e.g. only with stereotyped attributes). The stereotyped class is no longer a class, but a conceptual entity used in database modeling with all its properties. There is also a transformation defined which takes a well-formed «ERD_Entity» stereotyped class and transforms it into (for example) database table – which is also a class stereotyped with different stereotype and with different semantics.

6 Usage and creation guidelines

Based on the results of the industrial case study on MDA presented in [3, 12] a process of creation of transformational stereotypes can be proposed in Fig. 4. The process can be accompanied with a template for stereotype definition (documentation).

6.1 Process for creating the stereotypes

The process is presented in Fig. 4 using the SPEM notation (Software Process Engineering Metamodel [24]) for specifying software development processes.

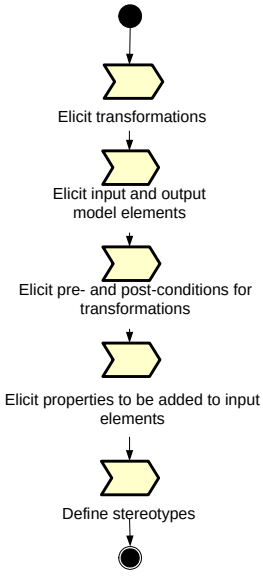


Fig. 4. Process of creating transformational stereotypes

The process has to start with eliciting transformations which are to be supported by transformational stereotypes. After this task is completed the input and output model elements need to be found since the input elements are to be used as base elements for

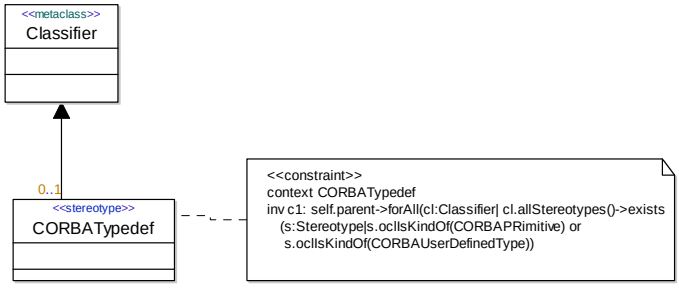
stereotypes. After the identification of elements, the pre- and post-conditions for transformations need to be specified since the pre-conditions are to be used as invariants for stereotypes for input elements. The post-conditions might be used as invariants for stereotypes for output elements (if these stereotypes are used). After specifying conditions the properties which are required for automatic transformations are elicited. The properties which are not in a set of standard properties of the input model elements are elicited and are used as a basis for tag definitions for stereotypes in the next step in which the stereotypes are defined and complemented with constraints elicited as pre-conditions for the transformations.

6.2 Generic template for stereotype documentation

The generic template for definition and documentation of transformational stereotypes is presented in Fig. 5 on the example of the «CORBATypedef» stereotype. Table 1 contains all possible elements which can be defined for transformational stereotypes. In this case they were provided to present the template although in specific cases the unused elements of the template can be omitted.

Table 1. Elements in definitions of transformational stereotypes of each type.

Element	Exomorphic		Endomorphic		
	Restrictive	Redefining	Descriptive	Restrictive	Redefining
Definition (diagram)	+	+	+	+	+
Name	+	+	+	+	+
Representation	+	+	+	+	+
Example result of transformation			+	+	+
Example excerpt of the source code	+	+			
Tag definitions	+	+	+	+	+
Constraints	+	+		+	+
Code generation template	+	+			
Mapping description			+	+	+
Transformation name	+	+	+	+	+



Name & representation	«CORBATypedef»	
Description	The stereotypes allow modeling an *IDL typedef* construct which is used for providing aliases for datatypes.	
Example excerpt of the source code	<<CORBAPrimitive>> X ▲ <<CORBATypedef>> Y	typedef X Y;
Example result of transformation	n/a	n/a
Tag definitions	n/a	
Constraints	c1	The parent must be stereotyped as «CORBAPrimitive» or as a descendant of «CORBAUserDefinedType» context CORBATypedef inv c1: self.parent->forAll (cl:Classifier| cl.allStereotypes()->exists (s:Stereotype| s.oclIsKindOf(CORBAPRimitive) or s.oclIsKindOf(CORBAUserDefinedType))
Code generation template	typedef {*NameOfStereotypedElement*} {*NameOfParentOfTheElement*};	
Mapping description	Source element	Target element
	n/a	n/a
Transformation name	Code generation for C++ CORBA	

Fig. 5. Template for defining transformational stereotypes[2]

7 Conclusions

In this paper a specific role of stereotypes in UML designs was elaborated on - supporting model transformations. It was found that there are certain properties of stereotypes dedicated to play this role. The properties are related to different functions which the transformational stereotypes fulfill – adding details and simplifying transformation code. The classification of the stereotype to an appropriate category pro-

[2] For the sake of clarity of the example we restrict ourselves to only one constraint of this stereotype.

vides creators of the stereotype with guidelines on how the stereotype should be implemented to increase its quality.

As the stereotypes are crucial for automating the process of transforming UML models (c.f. [3]) their effective creation should increase the overall quality of the model transformations. The proper definition of these stereotypes leads to the decrease of complexity of model transformations thus increasing such quality attributes as maintainability and correctness (as defined by McCall – c.f. [25]). Furthermore the proper separation of concerns – using well-formedness rules instead of pre- and post-conditions – increases such quality attributes as usability.

The intention of the analysis of stereotypes used in practice was to abstract common properties of the stereotypes which are already used in industry and which are known to be appropriately defined. Abstracting the properties allows transferring them into organization willing to reuse good practices from others and thus increase their competence. The properties can be used in a lightweight process for assessing quality of stereotypes as outlined in [2].

The guidelines for creating transformational stereotypes are intended to be simple since they are to be used in the course of lightweight processes for language customization – especially suitable for industrial needs. Therefore the initial overhead for learning them and following them might be minimized although there certainly is an initial overhead cost in terms of creating the appropriate stereotypes and assessing their quality. This overhead, however, introduces the return of investment in a longer run within the project since the proper stereotypes reduce the complexity of the transformations and thus reduce their development time and increase their maintainability.

We are currently working on a tool support for Telelogic Tau G2 which would enable effective documentation and guidance in creation of transformational stereotypes.

References

1. Object management Group, "UML 2.0 Superstructure Specification", Object Management Group, 2004, www.omg.org, last accessed 2005-03-31.
2. Staron M. and Kuzniarz L., "Properties of Stereotypes from the Perspective of Their Roles in Designs", In the Proceedings of 8th International Conference on Model Driven Engineering Languages and Systems, Montego Bay, Jamaica, 2005, pp. accepted for publication.
3. Staron M., Kuzniarz L., and Wallin L., "A Case Study on Transformation Focused Industrial MDA Realization", In the Proceedings of 3rd UML Workshop in Software Model Engineering, Lisbon, Portugal, 2004, pp.
4. Warmer J. B. and Kleppe A. G., "The Object Constraint Language : Getting Your Models Ready for MDA". Boston, MA, Addison-Wesley, 2003.
5. Kleppe A. G., Warmer J. B., and Bast W., "MDA Explained : The Model Driven Architecture : Practice and Promise". Boston, Addison-Wesley, 2003.
6. Miller J. and Mukerji J., "MDA Guide", Object Management Group, 2003, http://www.omg.org/mda/, last accessed 2004-01-10.
7. Gogolla M. and Henderson-Sellers B., "Analysis of UML Stereotypes within the UML Metamodel", In the Proceedings of The 5th International Conference «UML» 2002 - Unified Modeling Language. Model Engineering, Concepts, and Tools, Dresden, Germany, 2002, pp. 84-99.

8. Atkinson C. and Kühne T., "Rearchitecting the UML Infrastructure", *ACM Transactions on Modeling and Computer Simulation*, vol. 12, 2002, pp. 290-321.
9. Atkinson C. and Kühne T., "The Role of Metamodeling in MDA", In the Proceedings of Workshop in Software Model Engineering, Dresden, Germany, 2002, pp. Np.
10. Atkinson C. and Kühne T., "Model-Driven Development: A Metamodeling Foundation", *IEEE Software*, vol. 20, 2003, pp. 36-41.
11. 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.
12. Staron M., Kuzniarz L., and Wallin L., "A Case Study on Industrial MDA Realization - Determinants of Effectiveness", *Nordic Journal of Computing*, vol. 11, 2004, pp. 254-278.
13. Wirfs-Brock R., "Stereotyping: A Technique for Characterizing Objects and Their Interactions", *Object Magazine*, vol. 3, 1993, pp. 50-3.
14. Kuzniarz L., Staron M., and Wohlin C., "An Empirical Study on Using Stereotypes to Improve Understanding of UML Models", In the Proceedings of The 12th International Workshop on Program Comprehension, Bari, Italy, 2004, pp. 14-23.
15. Staron M., Kuzniarz L., and Wohlin C., "An Industrial Replication of an Empirical Study on Using Stereotypes to Improve Understanding of UML Models", In the Proceedings of Software Engineering Research and Practice in Sweden, Linköping, Sweden, 2004, pp. 53-62.
16. Object Management Group, "Unified Modeling Language Specification V. 1.5", Object Management Group, 2003, www.omg.org, last accessed 2003-10-01.
17. 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.
18. 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. 262-270.
19. Pollet D., Vojtisek D., and Jézéquel J.-M., "OCL as a Core Transformation Language", In the Proceedings of WITUML: Workshop on Integration and Transformation of UML models, Málaga, Spain, 2002, pp.
20. Kuzniarz L. and Staron M., "Improving Persistency Modeling in UML-Based Software Development Process", In the Proceedings of Software Engineering and Applications, Marina del Rey, CA, 2003, pp.
21. 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.
22. 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.
23. 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.
24. Object Management Group, "Software Process Engineering Metamodel Specification", OMG, 2005, www.omg.org, last accessed 2005-05-01.
25. McCall J. A., "Quality Factors", in *Software Engineering Encyclopedia*, vol. 2, J. Marciniak, Ed. NY: Wiley, 1994, pp. 958-971.