Proc. Estonian Acad. Sci. Eng., 2002, **8**, 1, 3–17

Techniques of combination of metamodel extensions

Alar Raabe

Computer Engineering Department, Tallinn Technical University, Raja 15, 12618 Tallinn, Estonia; alar@aetec.ee

Received 22 May 2001, in revised form 9 November 2001

Abstract. Extensions to the generic analysis and design metamodels (for example, UML metamodels) are needed to support an effective analysis and design process. Extended analysis metamodels that embody domain knowledge, and extended design metamodels which embody architecture-specific knowledge, can be used in software engineering as a guiding framework. Therefore it is necessary to combine several metamodel extensions. In the article solutions for combining several metamodel extensions and for the interoperability of metamodel extensions are presented. We also discuss how model transformations can use metamodel extensions.

Key words: metamodelling, metamodels, metamodel extensions, UML.

1. INTRODUCTION

Generic analysis and design metamodels do not help engineers during the analysis and design process. To achieve higher effectiveness of the analysis process, a more specific conceptual framework that embodies domain-specific knowledge is needed. The need for such extensions to generic metamodels has resulted in a number of efforts to develop domain-specific metamodels for business modelling (for example, Object Management Group (OMG) UML profile for Business Modelling [1] and Eriksson-Penker Business Extensions for the UML [2]), for the analysis and description of enterprise-wide data structures and conversions between them (OMG Common Data Warehouse Metamodel – OMG CWM [3]), and for modelling business workflows (Workflow Management Coalition Metamodel [4]).

When convergent engineering principles of software design [5] are applied, then analysis and design will produce artefacts that are easily mapped into the implementation. Automatic generation of implementation is possible if the used analysis and design metamodels are rich enough to involve needed technical details [6]. Generic metamodels of object-oriented analysis and design do not provide direct concepts for modelling domain-specific business structures, processes and products. Therefore in practical applications it is needed to extend the generic metamodels.

This problem is dealt with in the Generic Modeling Environment (GME 2000) [7,8]. GME 2000 uses three additional model operators to support model composition: equivalence of classes, interface inheritance and implementation inheritance of models. The equivalence operator constructs a union of two different classes. The new model inheritance operators define fixed selection criteria for model elements which are taken from the source model by one of these operators. Interface inheritance takes all the associations and compositions where source model element is in the role of the contained one. Implementation inheritance takes all the attributes and compositions, where source model element is in the role of the container.

Our experience shows that in practical applications it is necessary to allow more complex selection criteria for model elements which are inherited from the base models.

Since it is unlikely that for a given domain and implementation architecture only a single suitable metamodel extension exists, in practice metamodel extensions must be combined. These metamodel extensions are made usually by different parties, and therefore the methods used for metamodel extensions must ensure compatibility.

Problems that rise during the combination of metamodel extensions, analysed in the article, are the following:

– syntactic and semantic conflicts between elements from different metamodel extensions;

– overload of the resultant metamodel;

– difficulties to change the combined metamodel extensions afterwards.

Work presented in this article has been done in the context of a product-line architecture [9,10] for insurance applications, that applies principles of convergent engineering or model driven approach to software production [5,11], and has been developed under the guidance of the author.

Because insurance as an example of the problem domain is sufficiently complex, we assume that techniques efficient in this domain would be applicable to other domains.

2. METAMODEL EXTENSIONS

2.1. Definitions

The OMG Meta Object Facility (MOF) and the UML together form a four-layer metamodelling architecture described in [12], where metamodels are in the second layer, above the model and actual data.

There are several metamodel definitions in the literature [12,13]. In this article we shall use the following definitions for the metamodel and metamodel extension:

– *metamodel* is a model that defines the language for describing a model;

– *metamodel extension* is a set of additional or changed metamodel elements that together with the original metamodel form the language for describing the models which can capture more information than models that correspond to the original metamodel (called a base metamodel);

– *metamodel extension mechanism* is a set of model operations which, when applied to the metamodel, create an extended metamodel from the base metamodel;

– *combination of metamodel extensions* is a simultaneous application of metamodel extension mechanisms which create different metamodel extensions of the same base metamodel.

Because the UML [1] is the most widely used modelling language today, established as a single standardized modelling language, in this article we shall discuss the metamodel extension mechanisms in the context of UML.

2.2. Metamodel extension mechanisms in the UML

The metamodel of the UML can be extended either implicitly, using the extension mechanisms built inside the UML, or explicitly, using the MOF [12].

Implicit extension mechanism, profiles, impose restrictions on the possible modification of the UML metamodel. When extending the UML metamodel explicitly using the MOF, such restrictions do not apply. In principle, any metamodel can be defined and registered in the MOF. Therefore every profile definition can be expressed as a MOF metamodel, but not all MOF metamodels that extend the UML metamodel can be expressed as a proper UML profile.

2.2.1. Implicit metamodel extensions – UML profiles

The metamodel of the UML can be implicitly extended via stereotypes, tagged values, and constraints.

Stereotypes, the main mechanism of the implicit UML metamodel extension, are new metamodel elements which are subtypes of the existing metamodel elements.

In the proposal for the UML v 1.4 [14], the definition of the stereotype has been changed. Instead of using a tagged value for representing the relationship between a metamodel element and the stereotype, an association with the stereotype “stereotype” is used. Still, it is not clear whether it is possible to use the specialization relationship between the stereotypes themselves.

Tagged values are properties which can be attached to any model element. They form a restricted extension to the metaattributes of the metaclasses of the UML metamodel. If tagged values are attached to a stereotype, all the model elements conforming to this stereotype should have these tagged values.

Constraints are restrictions imposed upon a set of model elements which cannot be expressed via the UML notation and are therefore represented as text expressions. If constraints are attached to a stereotype, all the model elements conforming to this stereotype must conform to all the connected constraints.

The consistent set represented as a package of implicit UML metamodel extensions (stereotypes, tagged values, and constraints) and standard model elements (for example, comments), is called a UML profile. A UML profile can also have a set of prerequisite profiles required to define the given profile. The main advantage of UML profiles is the support by all compliant "off-the-shelf" UML tools.

The main constraint when using UML profiles is that they can extend the UML metamodel only strictly additively. Therefore profiles cannot change the existing semantics of the UML, they can only extend it.

2.2.2. Explicit metamodel extensions using MOF

The metamodel of the UML can be extended via the MOF by explicitly adding new metaelements to the metamodel. In principle, any metamodel can be defined and registered in the MOF. Therefore every profile definition can be expressed as a MOF metamodel, but not all MOF metamodels that extend the UML metamodel can be expressed as a proper UML profile. Graphically, it is possible to use UML for representing the MOF models (UML metamodels).

The advantages of defining a MOF metamodel instead of a UML profile are as follows:

– it is possible to introduce completely new metamodel elements which cannot be subtyped from the existing UML metamodel elements;

– it is possible to introduce classification hierarchies of new metamodel elements;

– the structure of a metamodel extension is better represented;

– description of a model is easier, because the familiar UML notation can be used instead of different notations for metaelements as in the case of implicit metamodel extensions;

– it is possible to use the metamodels other than modelling tools and exchange the metamodels between different tools.

For the classifiers, which represent more primitive things in the model like classes and associations, it is useful to impose the constraint of additivity of changes through inheritance, which ensures the substitutability of child classifier instances with the parent classifier ones. When the classifier is extended through the specialization, all the changes done via inheritance in the child classifier must preserve the semantics of the parent classifier.

For models as packages which are also generalizable elements, but not classifiers, substitutability is not required and therefore it is possible to implement generalization not only by adding the features, but also by changing and removing them. When models and packages are extended through the specialization, changes done via inheritance in the child element can change the semantics of the parent element.

2.3. Combining metamodel extensions

As described above, several areas exist in the software development process where the metamodel extensions are useful. In principle, it is possible to construct an analysis and design metamodel which contains all the required extensions as a whole. In practice, this would be extremely difficult because of the following:

– different metamodel extensions needed and applicable during different activities in software construction are usually defined by different parties at different times, and it is often impossible to synchronize these efforts;

– different metamodel extensions used by different tools are determined by these tools and cannot be easily changed; furthermore, other considerations exist besides the supported metamodel extensions of tools that affect the choice of concrete tools;

– to support the transfer of the software to a different base technology, it may be necessary to change the metamodel extensions, specific to the given base technology; thus there is a need for recombination of the metamodel extensions.

As shown in Fig. 1, transformations between different models (M_i), which possibly use different metamodel extensions, form another application where we need to combine source and target metamodels (MM_i) and their extensions to represent the transformation rules (which need to access concepts from both metamodels).

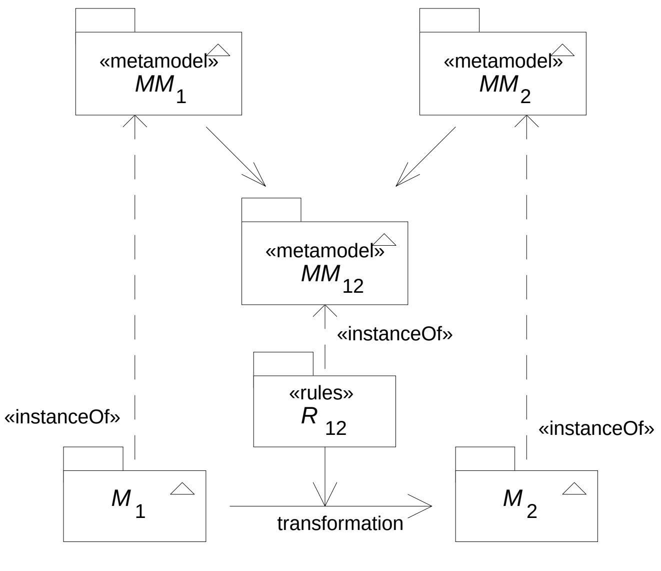

Fig. 1. Need for combined metamodels for model transformations.

2.4. Combining metamodel extensions in the UML

Several ways exist to combine metamodel extensions in the UML. In the case of implicit metamodel extensions or profiles, it is possible to apply several profiles to the same model. In the case of explicit metamodel extensions, it is possible to apply several mechanisms that are applicable to combine MOF models [12] because in the OMG four-layer metamodelling architecture MOF is used to represent the UML metamodel. Because the UML metamodel is also defined in the UML itself, alternatively it would be possible to apply mechanisms that are applicable to combine models in the UML.

2.4.1. Combining implicit metamodel extensions in the UML

Next we shall examine the techniques that can be used when combining meta-model extensions in the UML. These techniques are based on the assumption that the UML metamodel is defined in the UML itself and, as such, metamodel extensions can be combined using the model-combination mechanisms of the UML.

The UML allows using several profiles for the same model. Using the profile can be represented by the relationships between the models stereotyped as “appliedProfile” as proposed in the UML v 1.4 [14]. An example of graphical representation of the combination of multiple profiles is shown in Fig. 2.

The current version of the UML [1] does not allow multiple stereotypes for model elements, but in the proposed new version of the UML [14], the model can contain elements which are stereotyped with multiple stereotypes. This makes the creation of artificial stereotypes, which allow combinations of stereotypes, unnecessary when combining profiles.

2.4.2. Combining explicit metamodel extensions in the UML

If the metamodel extensions that are used in the example are defined explicitly and represented as UML models, we can use containment, importing, or multiple inheritance of models. Because a model can be an instance of only one metamodel, we have to create an intermediate metamodel that combines the extensions represented by the metamodels we want to use.

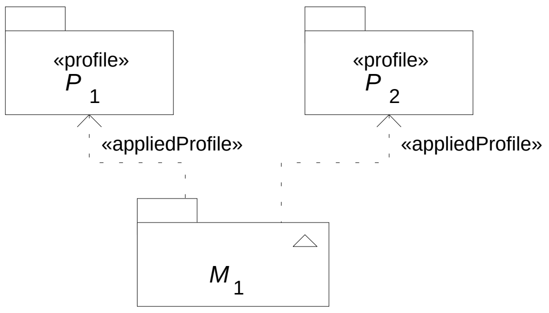

Fig. 2. Combination of multiple UML profiles.

When using model containment to combine different extended metamodels, combined model elements are encapsulated and not visible to the combining model or to each other. To be able to “see” those elements in the combining model, they have to be either imported or the combining model should be a specialization of all the combined models. Therefore in practice containment is not suitable for combining the metamodel extensions.

When using model importing for combining different extended metamodels, the dependence on the stereotype “imports” in the UML describes an access permission, i.e., that an importing model imports all the elements with sufficient visibility from the supplier models, including elements of models imported by the supplier models that are given public visibility in the supplier.

Elements imported from other models extend the namespace of the combining model. To avoid name conflicts, it is possible to rename the imported elements one by one. By default, the imported elements are given private visibility in the importing model, which can be changed during renaming. Because it is not possible to build metamodel hierarchies with importing, it has only limited value as a mechanism for combining the metamodel extensions.

When using multiple inheritance of models to combine different extended metamodels (see Fig. 3), because a model can have generalizations to other models, it is possible to construct taxonomic hierarchies of models. The mechanism of constructing the description of a specific model out of more general models is inheritance, i.e., the public and protected elements owned or imported by more general models are also available to its children in more specific models, and they can be used similarly to any element owned or imported by the child models themselves.

Elements inherited from other models due to generalization retain their name and extend the namespace of the inheriting model. By default, inherited elements

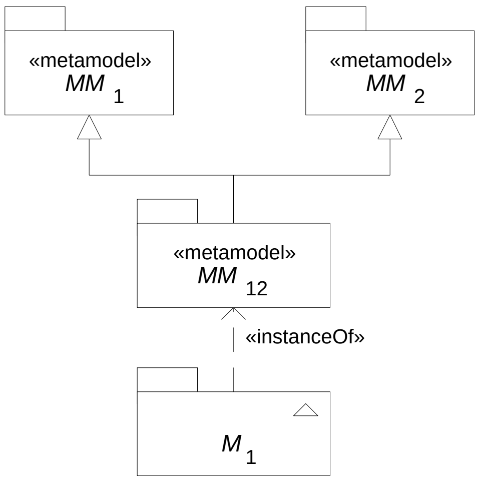

Fig. 3. Combination of multiple metamodels by multiple inheritance.

have the same visibility both in the child and the parent model. It is not possible to change the name or visibility of inherited elements in the inheriting model.

3. PROBLEMS OF COMBINING METAMODEL EXTENSIONS

Generic problems of combining metamodel extensions are the following:

– name conflicts;

– conflicting metamodel elements (conflicting features, relationships and constraints);

– cluttered resultant metamodel (because all the combination methods in UML are only additive);

– it is difficult to change the used metamodel extensions of the model.

Unclear semantics of a certain stereotype which is not the stereotype of a metamodel element but of some other stereotype, is a specific problem with implicit metamodel extensions – profiles in the UML.

A need for an intermediate metamodel which would combine several meta-models, is a specific problem with the explicit metamodel extensions. This stems from the semantics of the generalization relationship between the models and the metamodel, where the model cannot be an instance of several metamodels.

4. SOLUTIONS

To avoid the previously described problems, we propose to extend the semantics of importing the metamodel elements with filtering and massive renaming, and to extend the semantics of the metamodel inheritance with the three metamodel combination operations: override, substitution, and deferring.

Additionally we propose the usage of UML profiles similarly to the interfaces in object-oriented programming (OOP), to isolate the metamodel used for software development from the possible changes of metamodel extensions.

4.1. Resolving name conflicts and metamodel clutter

To avoid name conflicts when combining metamodel extensions explicitly, metamodel combination techniques like containment, importing, and inheritance, should allow massive renaming of elements. The present UML mechanism for importing elements from other models allows only renaming element by element.

To avoid name conflicts when combining several profiles, a profile should act as a namespace for stereotypes and tags. A possible solution to name conflicts is also a global name registry similar to the Internet domain name registries. To resolve the metamodel clutter, we propose to extend the semantics of metamodel importing with *filtering* and *massive renaming*.

Graphical representation of filtering and massive renaming is shown in Fig. 4.

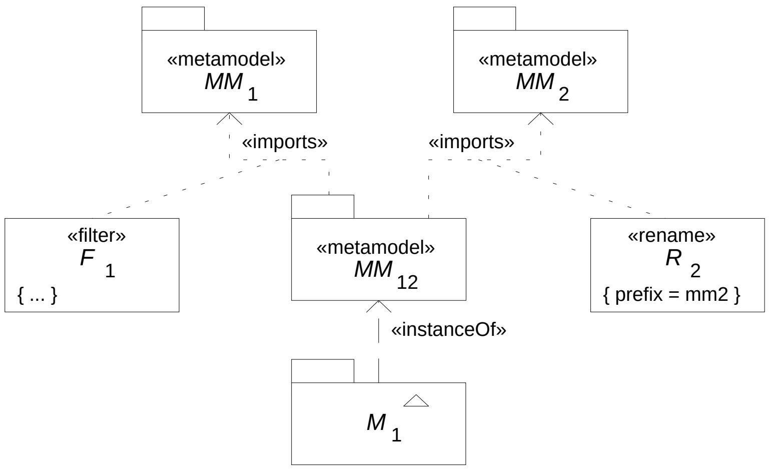

Fig. 4. Extended semantics of the metamodel combination via import.

Figure 5 presents an example of using the mechanisms described to combine two examples of metamodel extensions, usable in the domain of insurance software: design metamodel extension representing temporal concepts and implementation metamodel extension representing Enterprise JavaBeans (EJB) [15] specific concepts, so that from the first metamodel extension only extended classes and all base elements are imported (filter "ExtendedClassesAndBase"), and from second metamodel extension only extended elements are imported (filter "OnlyExtended") and their names are prefixed with "EJB". The filters "ExtendedClassesAndBase" and "OnlyExtended" contain the constraint given in

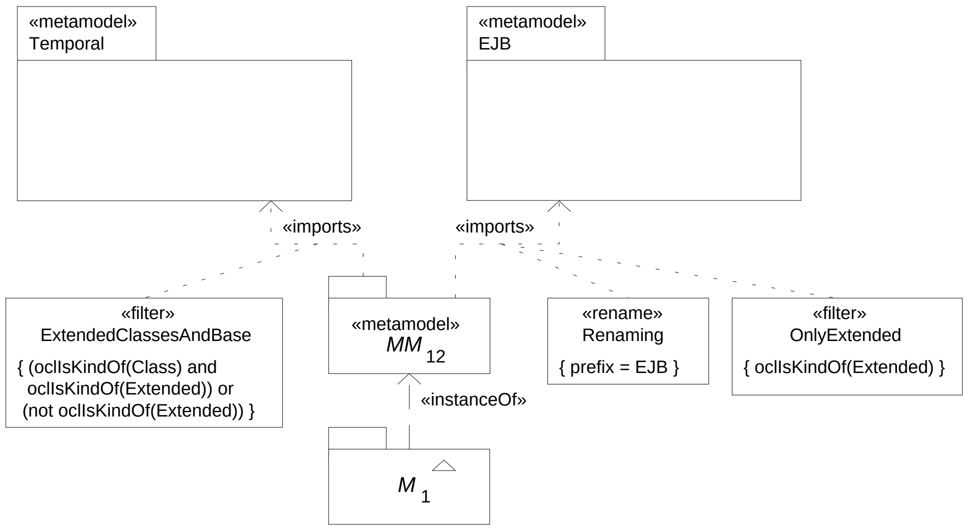

Fig. 5. Example of metamodel combination via import.

the object-constraint language (OCL), which describes the set of metamodel elements in the source metamodel which are imported into the target metamodel.

4.2. Resolving conflicts between metamodel elements

By definition, in the UML model inheritance semantics is such that all public or protected elements, which are owned or imported by the ancestor, are also available in the specialized model under the same name and interrelated as in the ancestor. The inheritance of models in the UML is similar to the inheritance of other classifiers. Because the nature of models is different from the classifiers, we propose to relax the substitutability requirement of models, and to solve the problem of possible conflicts between metamodel elements we propose to extend the semantics of inheritance for the metamodel combination with override, replace, and deferring operations.

When a metamodel element is *overriden*, the metamodel element in the ancestor is masked by the metamodel element in the child. This mechanism is analogous to the concept of overriding class features in the OOPL.

When a metamodel element is *replaced*, then the metamodel element in the ancestor is replaced by the metamodel element in the child (for instantiations of the child).

When a metamodel element is *deferred*, then the metamodel element in the ancestor is removed from the child (or suppressed in the child).

Graphical representations of the described extensions of the metamodel inheritance are shown in Fig. 6. The defer operation “D_1” contains the constraint given in the OCL, which describes the set of deferred model elements (elements of the parent metamodel which are not inherited to the child metamodel). Override and replace operations are denoted by the corresponding stereotypes on the metamodel elements of the child metamodel which override or replace the parent metamodel elements.

4.3. Using profiles as interfaces to metamodel extensions

To change the metamodel extensions used in the model afterwards, we propose to use implicit metamodel extensions or profiles as interfaces to the different, implicit, or explicit metamodel extensions. Later, they would play the role of implementation for the former.

Figure 7 describes a situation, where we have two different metamodel extensions, which we want to make changeable without affecting the model M_1. In this case, both profiles P_1 and P_2 will contain exactly the same set of stereotypes which contain the same tags, defined on different metamodels.

Here the dependence on the stereotype “extends” describes the relationship between the implicit metamodel extension and the metamodel.

The same technique can be used to prepare the model to be used by the future metamodel extension as shown in Fig. 8.

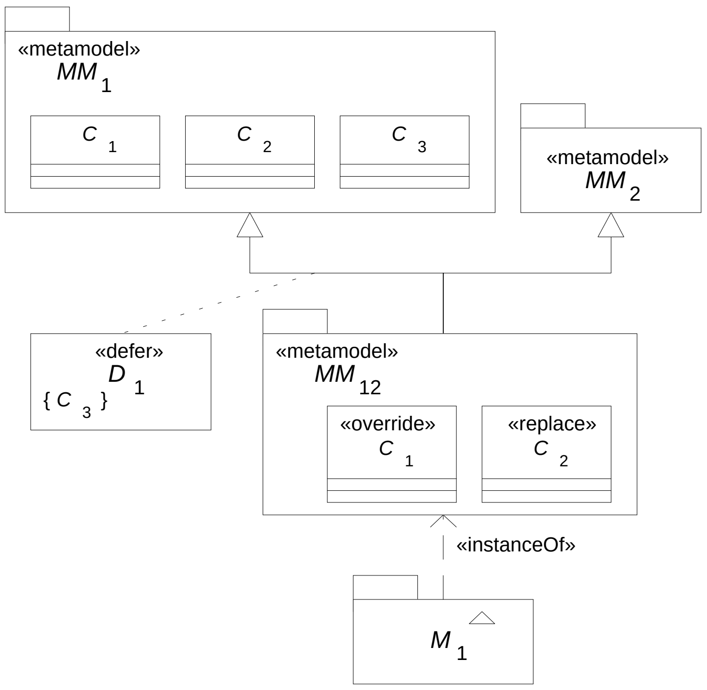

Fig. 6. Extended semantics of the metamodel combination via inheritance.

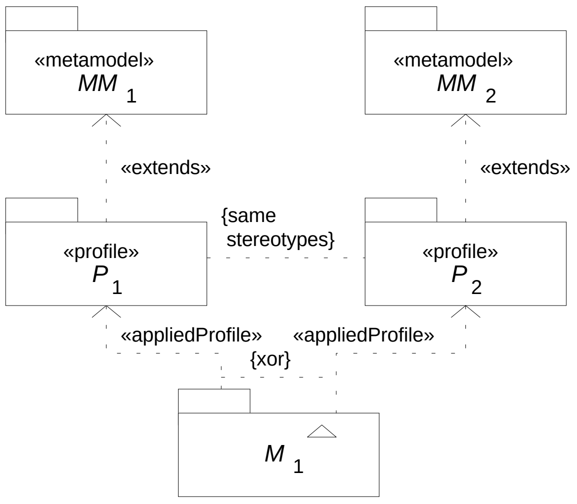

Fig. 7. Using profiles as interfaces to metamodel extensions.

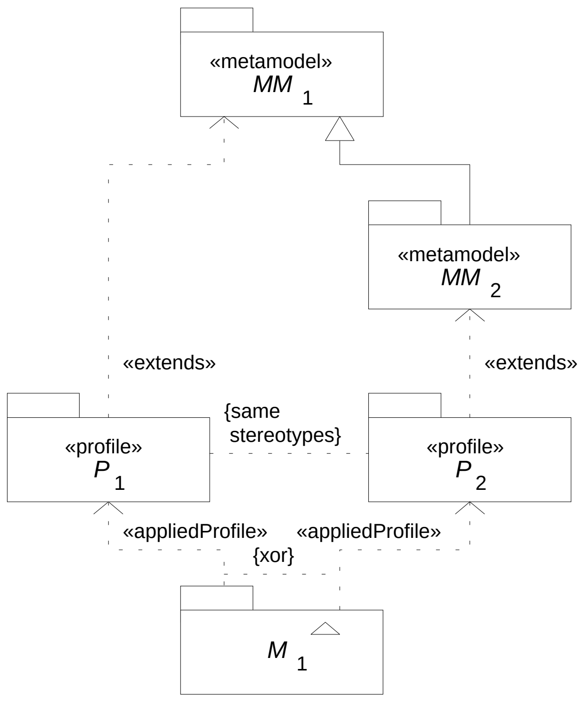

Fig. 8. Using profiles as interfaces to allow future use of a metamodel extension.

5. PRACTICAL APPLICATION

A practical application, where presented techniques of combination of metamodel extensions are used, was developed under the guidance of the author at the Progressive Financial Technologies Profit Ltd. (Finland) during 1995–2000, and is sold under the registered trademark Once&Done®.

Once&Done® is a product-line architecture to support the creation of insurance applications based on the convergent engineering principles and a special business model, which all belong to the same product-line and are based on the common object-oriented architecture.

Once&Done® product-line architecture consists of the following.

Models, which are metamodels that support object-oriented analysis of the insurance domain – the insurance-specific extension of the metamodel of the traditional object-oriented analysis, and analysis models of the insurance domain – used as a basis of analysis during a concrete insurance system creation, organized according to the main elements of the insurance domain (like party, policy, insurable, coverage) and according to the business lines of the insurance business (like property and casualty insurance, life insurance).

Framework, consisting of elements which implement technical (base) services for building object-oriented business software – an environment for business objects. This framework defines interfaces for implementing concrete business objects identified in the business domain and the concrete business functionality. Additionally, this framework contains a generic implementation of insurance domain models and the related insurance functionality.

Process, containing the description of steps and tasks required to create a member of the product-line, and based on the object-oriented paradigm. The goal is to support the creation of the insurance software based on the Once&Done® product-line architecture, maintaining the quality and predictability, identification of reusable elements, and the accountability (visibility) of the process.

Tools, containing facilities for using the framework and models according to the process to produce members of the product-line. A central tool is the Once&Done® Specification Environment (OD-SE), which implements the extended analysis and design metamodel and is an electronic environment supporting the process of building members of the product-line. The basic components of OD-SE are: the repository based on the extended metamodel, document management, discussion, and project management applications. Additionally, tools contain various generators which permit to generate concrete implementations of business objects based on the information in the OD-SE repository. OD-SE permits to connect several OD-SE repositories, enabling one to create members of the product-line by combining multiple existing models.

Once&Done® uses combination of metamodel extensions for construction of metamodels usable during the analysis, design and implementation phases of insurance software development.

Because the whole development cycle of software is based on the same model (according to convergent engineering principles) the software engineering process is simplified and the total amount of work is reduced, gaps between business processes and their supporting software are eliminated, and modifications to the business processes and the supporting software are easily coordinated.

Insurance products that can be composed of elementary parts to cover certain risks, involve complex business rules and form a large domain which must be separately modelled before the systems to support these products can be built. Combination of metamodel extensions suitable to describe insurance business processes and insurance products in Once&Done® allows description of business processes and insurance products as an integral part of the insurance systems model. When compared to traditional universal modelling methods, this diminishes the number of models that must be constructed, makes models smaller, and makes transformation from the analysis models to design models easier.

When changing the metamodel extensions that describe implementation mechanism (implementation architecture), it is possible to generate different implementations of the insurance system from the same model. This has been tested by changing the implementation architecture of the same insurance system from the client-server architecture for a fat client to three-tier server centric architecture for a thin client, without changing the insurance system model which was used to generate the implementation of the system.

6. CONCLUSIONS

Metamodelling and metamodel extensions to the existing industry standard metamodels (for example, the UML metamodel) provide a guiding framework for the analysis and design of software systems. The extended analysis metamodel that embodies domain-specific knowledge, guides an analyst during the analysis process, and an extended design metamodel that embodies architecture-specific knowledge, guides a designer during the design process.

Due to this, a need exists to combine different metamodel extensions.

To avoid name conflicts in metamodel extensions, we propose that the UML profile description be the name space for the stereotypes and the tags be defined in the given profile.

To allow easier combination of metamodels and metamodel extensions, a method of extending model import semantics in the UML with selective import and massive renaming is proposed. Additionally, to reduce the complexity of the resulting metamodel, an extension of the model inheritance semantics with the concepts of overriding, replacing, and deferring metamodel elements is proposed.

The problem of combination of metamodels and the proposed solution can be used also in the context of communicating agents. To successfully interpret the messages sent between agents, both agents must form a common metamodel of the universe of discourse. Proposed techniques can be applied to create this metamodel, based on the metamodels of involved agents.

Further research is necessary to find out how the change of the used metamodel extensions reflects on the semantics of the model and how it would be possible to ensure the model correctness after this kind of change.

ACKNOWLEDGEMENTS

Author wishes to acknowledge gratefully Progressive Financial Technologies Profit (Finland) and the Estonian Science Foundation (grant No. 4721) for their support. Author wishes also to thank Riina Putting, Boris Tamm and Kuldar Taveter for discussions on the subject and many useful suggestions for improving this paper.

REFERENCES

1. *Unified Modeling Language Specification, Version 1.3*. OMG Document, ftp://ftp.omg.org/pub/docs/formal/00-03-01.pdf.
2. Eriksson, H.-E. and Penker, M. *Business Modeling with UML, Business Patterns at Work*. John Wiley, New York, 2000.
3. *Common Warehouse Metamodel (CWM) Specification, Version 1.0*. OMG Document, ftp://ftp.omg.org/pub/docs/ad/01-02-01.pdf.

4. *The Workflow Reference Model*. WfMC Document TC00-1003, 1995 OMG, Workflow Management Facility Specification, OMG Document, ftp://ftp.omg.org/pub/docs/bom/97-11-18.pdf.
5. Taylor, D. A. *Business Engineering with Object Technology*. John Wiley, New York, 1995.
6. Melnikov, I. and Raabe, A. Expert system based computer aided software engineering (CASE) environment. In *Proc. 5. Symposium "Grundlagen und Anwendungen der Informatik", Wissenschaftliche Tagungen der Technischen Universität Karl-Marx-Stadt*, 1990, 6, 180–188.
7. Ledeczi, A., Volgyesi, P., and Karsai, G. Metamodel composition in the generic modeling environment. In *European Conference on Object-Oriented Programming (ECOOP'2001). Workshop on Adaptive Object-Models and Metamodeling Techniques*. Budapest, 2001, http://adaptiveobjectmodel.com/ECOOP2001/submissions/Ledeczi Vanderbilt E. COOP WS.pdf.
8. Ledeczi, A., Nordstrom, G., Karsai, G., Volgyesi, P., and Maroti, M. On metamodel composition. *IEEE CCA 2001*. Mexico City, 2001, http://www.isis.vanderbilt.edu/publications/archive/Ledeczi A 9 5 2001 On metamod.pdf.
9. Bass, L., Clements, P., and Kazman, R. *Software Architecture in Practice*. Addison-Wesley, Reading, Massachusetts, 1998.
10. Parnas, D. On the design and development of program families. *IEEE Trans. Softw. Eng.*, 1976, **2**, 1–9.
11. Abbott, B., Bapty, T., Biegl, C., Karsai, G., and Sztipanovits, J. Model-based software synthesis. *IEEE Software*, 1993, **10**, 42–52.
12. *Meta Object Facility (MOF) Specification, Version 1.3*. OMG Document, ftp://ftp.omg.org/pub/docs/formal/00-04-03.pdf.
13. Rumbaugh, J., Jacobson, I., and Booch, G. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, Massachusetts, 1999.
14. *Unified Modeling Language Specification, Version 1.4, Beta R1*. OMG Document, ftp://ftp.omg.org/pub/docs/ad/00-11-01.pdf.
15. *Enterprise JavaBeans Specification, v1.1*. Sun Microsystems Inc., 1999.

Metamudelite laiendite kombineerimine

Alar Raabe

Metamudelid toetavad süsteemide analüüsi ja projekteerimist. Laiendatud metamudeleid kasutatakse tarkvaraarendustes juhtiva karkassina. Artiklis on käsitletud metamudelite laiendite kombineerimise ning nende sobivuse tagamise võimalusi. Samuti on vaadeldud metamudelite laienduste kasutamist mudeli teisendamisel.