A Lightweight Approach for Domain-Specific Modeling Languages Design

Sylvain Robert, Sébastien Gérard, François Terrier
Laboratory of Model driven engineering for embedded systems
CEA LIST
Gif-sur-Yvette, F-91191 France
firstname.lastname@cea.fr

François Lagarde
Défense Sécurité R&D
SAGEM
Massy, F-91300 France
francois.lagarde@sagem.com

***Abstract*—Off-the-shelves general purpose modeling languages cannot obviously cover the whole range of needs that can be encountered in current systems design. Therefore, putting efficiently Model-Driven Engineering into practice involves designing specific modeling languages. The goal is to cover in a more suitable manner a particular application domain (e.g. automotive) or specific concerns (e.g. hardware modeling) or even to focus on a given class of practitioners. In this respect, two design approaches are generally opposed which respectively propose to define domain-specific modeling languages from scratch or to customize an existing general-purpose language. This paper focuses on the latter approach and claims that UML profiles do provide handy and powerful mechanisms to design domain-specific modeling languages but are penalized by lacks of methodological guidelines and tool support. To cope with these lacks, a profile design approach is introduced, which includes a methodological framework to structure profiles design process and tool support to partly automate this process.**

Model-Driven Engineering; UML; profiles; Domain-Specific Modeling Languages

I. INTRODUCTION

The MDA [2] initiative of the OMG (Object Management Group) has described the baselines for design approaches promoting models at the forefront of the development process. The objective is not only to use models as preliminary design artifacts or as communication support among development process' stakeholders, but to use models as an active driver of the design all along the development cycle, i.e. from requirements authoring to system implementation. The advantages of so-called model-driven approaches have been assessed several times (e.g. in [5]): models enable for more abstraction, more reusability, more automation and, this way, for less time and money to bring the system to the market, while keeping the same quality level. For these reasons, MDA is now well-established in the software engineering field and applied in a lot of related application domains, even in those that require fine-grained customization, like real-time embedded systems [1]. In a few words, MDA enforces a layered modeling process in order to raise abstraction level: instead of only one model, several models, iteratively and consistently refined from the most abstract to the most detailed, will be designed for each system. MDA emphasizes the need for multiple models, dedicated to specific viewpoints and / or levels of abstraction in order to fragment concerns and decrease design complexity. As such, the MDA approach is particularly suited to the UML or any equivalent large, general-purpose, multi-diagram modeling language.

On the other hand, some authors [3-4] consider that the UML is so large and targeting such a wide range of concerns that it is not only barely usable by non-UML experts, but also too rigid to really suit specific domains needs. Their opinion is that "model-driven development is domain-specific" [4]. Where the UML possibly targets all kinds of software engineering applications, DSMLs (Domain-Specific Modeling Languages) proponents prefer to focus on small, limited domains, worked by a relatively small number of developers. Among the advantages that this approach is said to bring [3], one is especially relevant: DSML approaches foster to focus on the concepts of the considered domain. Consequently, resulting models fit better to the actual system. This also implies that designers who are already acquainted with the domain will use more intuitively the language. However, these approaches have obvious drawbacks, among which the main one is the additional effort needed to design modeling languages from scratch.

As can be seen, UML and DSMLs have both valuable features…so which is the best to choose?

This paper claims that the good answer is probably not to choose. Actually, profiles provide a convenient way to design DSMLs based on the UML, thus building upon both above approaches. Profiles enable designing a wide variety of languages, from basic ones that only make minor (syntactical or semantic) adaptations to the base metamodel to more complex and exhaustive DSMLs. At first, the paper briefly introduces UML profiles and highlights some limitations in the current support for their design. Then, a profile design support framework is introduced to tackle these limitations. The aim is to enable full exploitation of UML profiles mechanisms in a user-friendly way.

The paper is structured as follows: Section II provides a few reminders about UML profiles and provides the outcomes of a study performed on a selection of OMG standard profiles. This study demonstrated that profiles are unevenly designed and highlighted the need for a better profile design support. Then, Section III and IV introduce our profile design methodology and framework. These latter provide sound methodological guidelines (Section III) and systematize and partly automate the profile design process (Section IV). At last, Section V elaborates on other related research works before concluding in Section VI.

II. PROFILE-BASED DSMLS

In this section, a few reminders about UML profiles are provided, before presenting the results of a study which shows that even standard OMG profiles are unevenly designed.

A. UML Profiles Basics

In order to fully specify a modelling language, at least three elements are required [6,7,23]. Firstly, the abstract syntax specifies the valid constructions of the language. In the scope of a DSML, the abstract syntax is usually specified with a metamodel (e.g. using the MOF [11]). Then, the concrete syntax describes the concrete representations of the concepts defined in the metamodel. It is also called "surface language" or "notation". At last, the semantics provides the meaning of modelling language elements. This aspect is usually closely related to the abstract syntax. The semantics may be formal, but it is not mandatory (for instance, the UML specifies it using natural language).

Defining a DSML from scratch may therefore, depending on its size and complexity, require some consequent efforts. The intent behind UML profiles is to decrease this effort by enabling reuse of (or part of) the UML metamodel and concrete syntax. This approach is relevant since a lot of DSMLs end up with common features. For instance, all workflow modelling languages (like BPMN [8], UML activity diagrams [9], Windows Workflow Foundations [10]) share equivalent concepts and representations: activities, gateways, transitions, etc. More concretely, profiles are built-in extension mechanisms applicable to the UML. A profile contains a set of stereotypes, each of which represents an extension of one or more several UML metaclasses. A stereotype may introduce new meta-attributes, semantic specialization or extension (it may for instance fix a UML variation point), additional constraints as well as new notations to the extended metaclasses. At the metamodel level, a profile is simply a kind of UML package containing a set of stereotypes, each of which extends at least one UML metaclass. At the modelling level, stereotypes appear as annotations on extended modelling elements. If specified in the profile, the notation of the stereotyped modelling element may also be totally modified. The profiles features presented above (syntactical and semantic extensions based on stereotypes) may also be completed by additional mechanisms. For instance, stereotypes may be parameterized with properties (also called tagged values). Designers may also add constraints (e.g. written in OCL [20]) to specify how stereotypes shall be used to obtain well-formed models. At last, since stereotypes are UML classes, they can be related (inheritance and associations) to form more structured profiles.

B. Building "Good" UML Profiles

From the example of the previous section, the reader could infer that building a profile is a breeze. Actually, it can be quite complex, especially when profiles are large, like e.g., the MARTE UML profile (the OMG reference UML profile for real-time embedded application modelling) [12], which introduces about 150 stereotypes. Building a profile is an intricate task, which requires to be properly mastered in order to avoid ill-formed or awkward results. This is even tangible in the scope of standard OMG profiles, as shown by a systematic study we have conducted in [13] on five standard UML profiles: SPEM [15], SysML [16], MARTE [12], UPDM [17] and SPT [18].

In order to perform a quantitative evaluation of the considered profiles, a set of relevant metrics have been defined. The first is the Average Number Location Application (ANLA) which gives, for a profile, the average number of UML elements its stereotypes may apply to:

$$anla = \frac{1}{n}\sum_{i=1}^{n} Mi$$

where N is the number of stereotypes in the profile and Mi is the number of metaclasses the ith stereotype may apply to.

Actually, a stereotype may be applied to any concrete subclass of the metaclass it extends. As a result, the ANLA will be higher if extended metaclasses are closer to the root element of the metamodel referenced by the profile than to its leaves. By way of illustration, a stereotype which extends the UML metaclass Element may be potentially applied to 199 metaclasses.

The second metric is the Average Leaf Depth of Inheritance Tree (ALDIT), which represents the average length of stereotypes inheritance trees in the profile:

$$aldit = \frac{1}{n}\sum_{i=1}^{n} Li$$

where N is the number of stereotypes in the profil and Li is the number of parent stereotypes of the ith stereotype. If the stereotype does not specialize any parent stereotype, Li = 0.

At last, the Average Stereotype With Attributes (ASWA) metric gives the average number of stereotypes having attributes:

$$aswa = \frac{1}{n}\sum_{i=1}^{n} Ai$$

where N is the number of stereotypes in the profile and Ai = 1 if the ith stereotype owns 1 or more attributes, 0 else.

These three metrics are complementary to each other and, taken together, they give a good picture of the considered profile. The ANLA indicator gives a hint of how much the profile is dependant to the UML metamodel. A high ANLA implies that the profile extends metaclasses close to the root element of the metamodel, i.e. which semantics is more general. The resulting profile will not be strongly dependent of the UML metamodel. On the contrary, a low ANLA shows that the profile makes use of leaves metaclasses and therefore, that it is highly dependent from the UML semantics. The ALDIT and ASWA metrics have a different focus, since they enable characterizing profile features. Basically, they give an idea of the complexity of the considered profile. The ASWA metric also shows if the profile (or its stereotypes) may be parameterized by the users.

TABLE I. gives the results obtained when applying the above metrics to the five selected OMG standard profiles. Since it would have been particularly tedious to evaluate manually the indicators for each profile, we have developed a dedicated tool within the Eclipse platform to calculate automatically these indicators [13]. Note that the "Nb. St." row gives the number of stereotypes defined the in the considered profile.

Several conclusions may be drawn from TABLE I. . First, it is obvious that there are a lot of disparities between the profiles. This is tangible when looking at profile sizes, but not only. The ANLA is eloquent if we consider the two extremes: the MARTE profile has 49 while the UPDM profile has 6.8. This means that UPDM make use mostly of UML leaves elements, while MARTE stereotypes extend more high-level metaclasses. This also means that MARTE stereotypes are more adaptable than UPDM ones since a stereotype which extends a high-level metaclass can be applied on all the concrete child metaclasses. The difference between the two ANLA values also indicates that MARTE is more independent from the UML. It is confirmed by the ALDIT and ASWA values which show that MARTE stereotypes have longer inheritance tree and own more attributes. This indicates that the MARTE profile defines a DSML which is more decoupled from the UML, with a more complex structure and with more added semantics. The other profiles follow more or less the same patterns but with less intensity: for instance, SPT is, as MARTE, more decoupled from the UML and more structured whereas SysML appears to have a simple structure, and to be more close to the UML.

These results show that objectives and outcomes may differ a lot when defining a profile. Profiles features will highly depend on the needs of designers and, most importantly, on their expertise. From our point of view a "good" profile: (i) results in a DSML which exhaustively covers the domain; (ii) relies on the UML metamodel while keeping a certain level of independence; (iii) is not too rigid, i.e. it may be parameterized to better suit user's need. The two last points are particularly crucial. In order to enlarge profiles audience, it is important to ensure that they do not restrict users' possibilities. The results obtained on the OMG standard profiles show that it is not yet true, since several profiles have low ANLA values. Actually, building such a profile requires a deep knowledge of the UML metamodel. Firstly, to properly choose the UML metaclasses to extend, but also to model profile structure (in terms of stereotypes relationships) and to define stereotypes attributes .If these steps are not mastered, the risk is to end up with a too rigid, or too large, inefficient profile. However, to our opinion, these obstacles could be overcome with a proper design support, as the one presented in the subsequent sections.

TABLE I. RESULTS OF METRICS APPLICATION TO THE MAIN OMG STANDARD PROFILES

	UPDM	SysML	MARTE	SPT	SPEM
Nb St.	165	39	149	56	57
ANLA	6.8	14	49	29	19
ALDIT	0.7	0.2	1.7	0.3	1.3
ASWA	0.17	0.43	0.72	0.6	0.15

III. A MODEL-DRIVEN UML PROFILE DESIGN PROCESS

This section presents a process aiming at structuring profile design activities. Section III.A outlines the motivations, while Section III.B describes the process itself.

A. *Motivations*

As highlighted in the previous section, the profile definition process should be well-structured and properly supported to avoid resulting in invalid or poor quality profiles. Unfortunately, this is not yet true, as showed by other (more qualitative) results of the study introduced in Section II.B. The most eloquent outcome is about the methodology, since even on this crucial aspect, no consensus seems to exist within the set of profiles studied. It is usually considered as a basic requirement to design a domain model (a UML-independent representation of the application domain dedicated to domain experts) in order to clearly separate application domain concerns from UML metamodelling concerns [14]. Despite this, the study shows that among the five analyzed profiles specifications, only three of them defined such a domain model. On several other aspects, some disparities have also been identified. For instance, it appears that no convention specify how constraints on stereotypes application shall be expressed (only three profile makes use of the OCL). Also, stereotype semantics is sometimes partially expressed (only SysML introduces several semantics for a single stereotype extending several metaclasses). This clearly states a lack of support for profile design, especially on the methodological side, hence our effort to define a methodological framework which could contribute to addressing these limitations.

B. *Overview of Profile Design Process*

To our opinion, the same factors shall be considered in a profile design as in a software development process:

- Actors: two kinds of actors have to be differentiated in a profile design process: domain experts who are familiar with domain concepts and technologies (e.g. automotive real-time software) and language engineering experts who bring (among others) their knowledge of metamodelling and of the UML.
- Evolution: as software, a DSML is bound to be modified when requirements change. It is then necessary to be able to take potential evolutions into account.
- Reuse: one of the main requirements of software engineering is to design for reuse. In the same way, DSMLs design processes shall facilitate reuse of knowledge and languages elements.

The distinction between the domain model and the profile is necessary with respect to the two kinds of actors identified above. Domain experts will be responsible for building a representation of the domain concepts in the domain model. On the other side, language engineering experts will use their knowledge of both metamodeling and the UML to map domain model concepts to the most appropriate UML elements and then, to design the most suitable profile according to domain model requirements. In addition, in order to foster reuse and facilitate evolutions, we draw our inspiration from the work performed on system families modelling [22] and we propose to consider not only the DSML being designed, but also its variants. This leads to design not a single profile, but instead a family of languages. Therefore, the resulting process includes three activities: Problem description (performed by domain experts), restriction and refinement (performed jointly by domain expert and language expert) and profile definition (performed by language expert). The process is schematized in Figure 1. using a SPEM-like syntax (dashed arrows link activities and their performers while plain arrows represent activities sequencing).

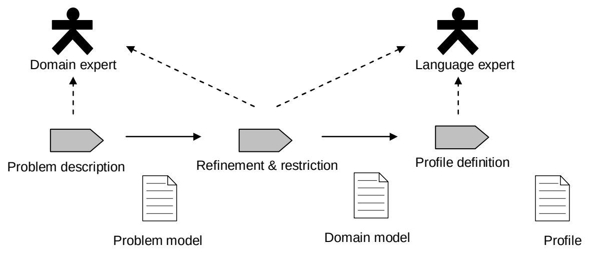


Figure 1. Overview of the Profile Design Process

The problem description activity is performed by domain experts. It consists in defining all domain concepts and their relationships corresponding to a family of languages. Therefore, it entails to identify those concepts that are invariant and those that may change depending on the profiles (i.e., the variation points). The problem description is expressed in a UML class diagrams.

The restriction and refinement step is performed jointly by domain experts and language experts. The aim is to select, among the concepts that are defined in the problem description those that apply to the profile under design (restriction). Within this activity, variation points are also fixed (refinement) and – potentially – some concepts may be refined. The outcome is a model describing only the concepts of the considered profile (the domain model as defined by B. Selic). This model is also expressed with a UML diagram.

The last activity deals with designing the profile itself. It includes three steps. Firstly, an incomplete profile is automatically generated from the domain model thanks to a set of predefined heuristics. Each class from the domain model is transformed to a stereotype and all relationships are temporarily kept intact. Then, the stereotypes are linked to the most relevant UML metaclasses. This step is critical and is entirely performed manually; only a deep knowledge of the UML metamodel will guarantee that the most adapted metaclasses are chosen. At last, the profile is modified – with some tool support - to ensure its correctness and for optimization purposes. The profile design activity is performed by metamodelling language engineering experts.

The first two activities (problem description and domain modelling) do require a good expertise of the domain but do not present any particular difficulties with respect to modelling. Actually, only a basic knowledge of UML class diagrams is required. However, this is not the same for profile definition: mapping stereotypes from the draft profile to UML metaclasses requires a good expertise of the UML. Moreover, this phase also entails to modify the profile in order to assess its correctness and to enhance its usability, i.e. to optimize it. In this respect, some modelling patterns may be identified which influence specific features of the resulting profile. The subsequent section describes these patterns and shows how to exploit them.

IV. PROFILE ASSESSMENT AND OPTIMIZATION

In this section, we describe more precisely the last activity of the process presented in Section III.B, i.e. profile definition. This activity aims firstly to ensure correctness of the profile. This entails to replace all constructions of the generated incomplete profile that are not valid. More precisely, the issue is to validate remaining stereotypes associations and to map them to profile-compliant constructions. Note that some relationships may be removed if they already exist at UML level. Then, the second objective of the activity is to optimize the profile: in our case, we have focused on reducing the total number of stereotypes. However, it is obvious that other optimization criteria shall be considered as part of our future works.

Several guidelines have been defined which help designers in reaching these two objectives. They are introduced in the remainder of this section.

A. Guidelines for associations validation and translation

At the domain model level, it is often necessary to express relationships between domain concepts. Designers usually express these relationships with UML associations. However, associations' semantics is not precise enough to express how stereotypes shall relate concretely at model-level. It is therefore necessary to translate associations to less ambiguous constructions when moving to the profile. More precisely, the issue is twofold. First, it is required to ensure that the associations introduced at domain-level are valid with respect of the UML metamodel. Then, when validity is assessed, a proper transformation has to be chosen among the several that are possible.

As far as validity is concerned, designers have to keep in mind that a profile shall not conflict semantically with the UML metamodel. A UML profile may extend the UML but cannot, by any way, modify it or introduce modelling constructions that contradict its initial semantics. In the case of associations, the question is thus whether such a relationship between two stereotypes is authorized with respect to the extended metaclasses relationships. Basically, two cases shall be distinguished. In the first one, the metaclasses extended by the stereotypes have no relationships. In this situation, the metamodel does not constrain in any way stereotypes relationships and any association between them is valid.

In the second one, there are one (or more) association(s) between the extended metaclasses. In this case, it is up to the designer to determine if it is necessary to redefine an association in the profile. The only way to do this is to consider closely the semantics of the association defined in the metamodel and to determine that it corresponds to the stereotypes' association one. If the answer is "no" without doubt, then the stereotypes' association is kept. If the answer is "yes", then other features shall be considered: association name, roles and multiplicities. If all these features are strictly equivalent in the two associations, then the stereotypes association may be removed since it would redundant with the one defined in the metamodel. However, if associations are semantically equivalent but differ on some of the aforementioned features (e.g. multiplicities), the best solution will be to keep the stereotypes' association but to indicate that it subsets/refines an existing metamodel association with appropriate tags ("subsets"/"refines"). Before going to the next step, note that the considered case only addressed stereotypes which extend a single metaclass. However, the approach extends quite well to more complex situations: if the stereotypes extend more than one metaclass, then the previously described analysis shall simply be performed on all extended metaclasses' associations.

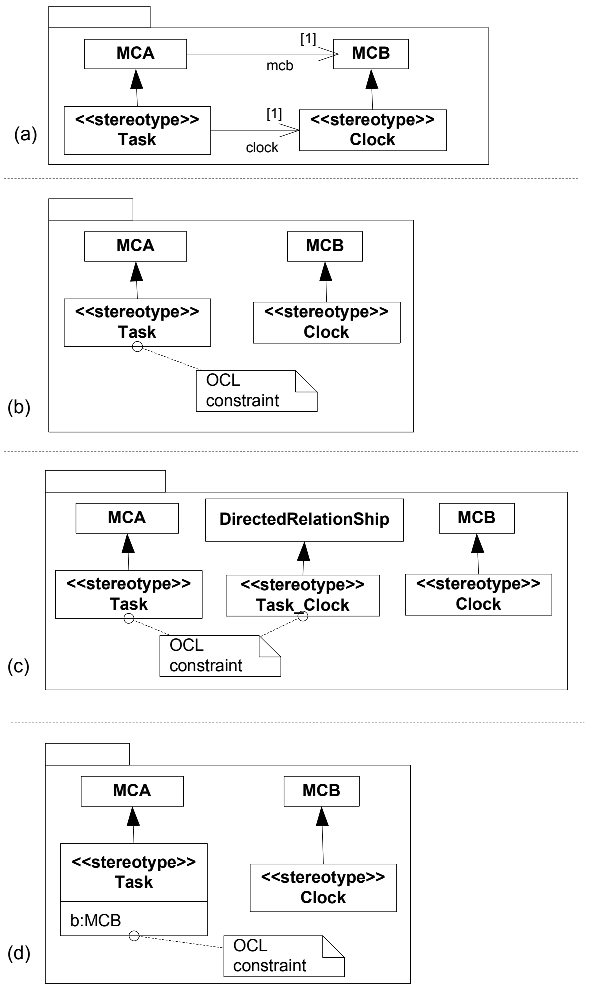


Figure 2. Possible translations of an association between stereotypes

Once all profile stereotypes' associations have been removed, modified or validated, the issue is to translate those that remain. Actually, it is undetermined how associations between stereotypes shall be represented at model-level; therefore, they have to be translated to less ambiguous constructions. Let us consider the basic example of Figure 2. a where a stereotype «Task» has an association to the «Clock» stereotype. These two stereotypes respectively extend the MCA and MCB metaclasses which are also related by an association.

In order to translate the stereotypes' association, the first possibility is simply to rely on the association which already exists between the base metaclasses of the stereotype (assuming that it is valid with respect to the expected semantics: see above). In this case, there is no additional attribute or stereotype to define. However, in order to ensure that the stereotyped elements will be properly related at model-level, an OCL constraint has to be added on the stereotype «Task» (Figure 2. b). This last solution is obviously the most compact. However, note that it apply only in the case when the two metaclasses are already associated (on the contrary to the subsequent strategies which apply in any case).

A second possibility will be to create a dedicated relationship to relate the two stereotyped elements at model-level. In this case (Figure 2. c), it will be required first to define the stereotyped relationship (the «Task_Clock» stereotype which extends the DirectedRelationShip UML metaclass). But it will also be necessary to add OCL constraints on both «Task» and «Task_Clock» stereotypes in order to guaranty, at model-level that: (i) all model elements stereotyped with «Task» will be related to a model element stereotyped with «Clock» (ii) that relationships stereotyped «Task_Clock» link only model elements respectively stereotyped «Clock» and «Task». This solution is the most relevant with respect to profile readability. However, it adds to profile size since it requires defining an additional stereotype.

The mast option is to define an attribute in the «Task» stereotype typed by the base metaclass of «Clock», as showed in Figure 2. d. In order to ensure that the attribute will have the right stereotype, it will also be necessary to attach an OCL constraint to the attribute (since OCL is a very verbose language, OCL constraints will not be detailed in the paper. Please refer to [13] for more details). This approach is relevant from the profile compactness point of view. However, profile readability is negatively impacted, since the relationship is only reified by the attribute

As we have seen, among the three possible translations, each offers specific advantages and drawbacks and none of them is ideal. The choice of the best will thus mainly depend on designers priorities.

B. Design patterns for profile optimization

As said earlier, many profile optimization patterns could be considered, but we have restricted our focus to cases where the number of stereotypes is decreased. A profile is indeed usually more readable if it is compact, i.e. if an effort to limit the number of stereotypes was made. There are actually situations (patterns) where stereotypes are defined but are not strictly necessary. The aim is thus to systematically identify these patterns and to replace them by more synthetic constructions.

For the moment, we have identified two of these patterns. The first one, named "lonely compound stereotype" pattern, is illustrated in Figure 3. a. A stereotype «TestCase» is introduced as a compound of a stereotype «TestContext». The stereotype «TestCase» does not bring any specific additional information since, firstly, it has no attributes and secondly, it does not participate into any other relationships. It may therefore be removed and replaced by a simple attribute of the stereotype «TestContext», as shown in Figure 3. b.

The second pattern, called the "superfluous child" stereotype pattern, is the one shown in Figure 3. c. A stereotype has one (or more) child stereotype(s) which again do not own any attributes and do not participate in other relationships. Therefore, these stereotypes may be removed and replaced by an enumerated attribute in the parent stereotype, as shown in Figure 3. d.

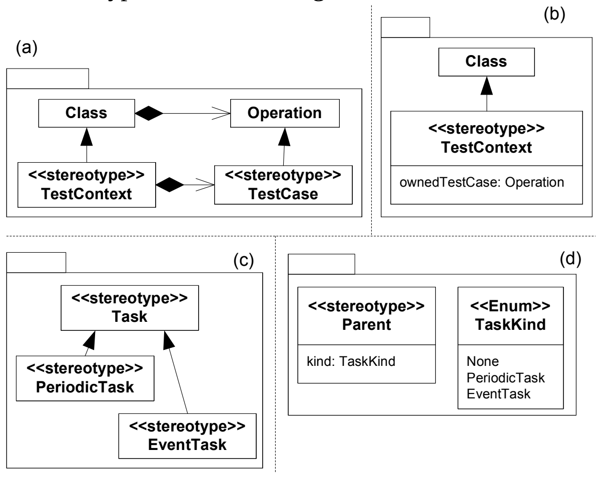
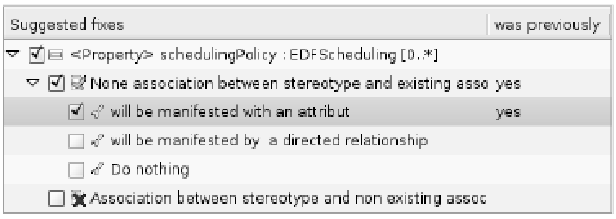


Figure 3. "Lonely compound" and "superfluous child" stereotype optimization patterns



```
Suggested fixes                                          was previously
▽ ☑ <Property> schedulingPolicy : EDFScheduling [0..*]
   ▽ ☑ None association between stereotype and existing asso   yes
        ☑ will be manifested with an attribut                   yes
        ☐ will be manifested by  a directed relationship
        ☐ Do nothing
     ☐ Association between stereotype and non existing assoc
```

Figure 4. Association translation wizard

C. *Tool support*

Whether it is about association translation or profile optimization, the final choice has to be made by the designer. Actually, transformation and optimization patterns introduced in Sections IV.A and IV.B take only into account pure modelling motivations. However, designers may have relevant motivations that go beyond this scope. For instance, compactness may not be the priority: in this case, the designer would certainly not choose to apply the optimization patterns introduced in Section IV.B.

Therefore, our tool support provides only decision-making support and does not perform the transformations arbitrarily. The tool, implemented within the Papyrus UML platform [19], is a wizard which analyzes input profiles and proposes to the designer possible modifications and optimization. Each time it detects a known pattern (e.g. the "lonely compound stereotype" pattern) in the analyzed profile, it displays all possible translations and the designer simply chooses his/her preferred one. Then the tool performs the chosen translation. By way of illustration, Figure 4 displays a part of the window that shows up in the case of association translation.

On top of profile optimization support, the tool also helps the modeler in the very first steps of profile design process. Once the domain model is built, it generates a skeleton of the profile (each metaclass is transformed into a stereotype and relationships are kept as they are) which may be used as a starting base to build the final profile.

V. Related Works

As mentioned earlier, to our knowledge, few research works have dealt with UML profiles design until now. We have already quoted the paper of B. Selic [14] which gave a first set of sound guidelines for profile definition and called for further work on the topic. In the scope of design profile practices analysis, we can mention too the work of Berner et al on stereotypes classification [21]. Our work on profile has also strong relationships with research works on (non UML-based) domain-specific modelling languages, like [3-4] and – of course – is "genealogically" related to the domain-specific languages research field [6,26]. In addition, since our research focuses on profile design support, it is closely related to all works that deal with profile-based modelling; especially those that tackle profile composition. Profile composition aims at determining ways to enable safe and consistent application of several profiles in the same model. On this topic, some results are already available: for instance, in [24] where the issue of composition is tackled through MARTE [12] and SysML [16] examples. At last, it is also worth mentioning [25] where an approach to defining inter-profiles constraints for safe multi-profiles application is introduced.

VI. Conclusion and Future Works

UML profiles provide powerful and flexible mechanisms for building DSMLs. However, a thorough quantitative and qualitative study of the main OMG standard profiles showed that profiles quality is too uneven and that profiles design is sometimes not mastered enough. This concern led us to propose a profile design methodology and an underlying framework to guide and support profile designers. They provide a description of the profile design process as well as some decision-making support tools, and semi-automated transformations to validate and optimize profiles.

However, the issue of profile design support remains open on several aspects. For instance, as far profile optimization is concerned, we plan to define additional optimization criteria and the corresponding set of transformation patterns. The aim would be to generate automatically an optimized profile, based on user-configured optimization criteria prioritization. Then, with respect to profile validation, it is also critical to study how stereotypes semantics can be validated against the semantics of extended metaclasses. This issue is especially critical in the cases where stereotypes extend high-level metaclasses (high ANLA values), since the semantics of all child metaclasses in the hierarchy have to be considered.

References

[1] F. Terrier and S. Gérard, Model-driven engineering and prototyping of real time embedded applications, in DIPES conference, Braga, Portugal, 2006.

[2] OMG, MDA Guide Version 1.0.1, June 2003.

[3] S. Cook et al, Domain-Specific Development with Visual Studio DSL Tools, Microsoft .net development series, 2007.

[4] S. Kelly and J.-P. Tolvanen, Domain-Specific Modeling, IEEE Computer Society, ISBN 978-0-470-03666-2, 2008.

[5] P. Baker, S. Loh, F. Weil, MDE in a large industrial context: the Motorola case study, in MODELS conference, Montego Bay, Jamaica, 2005.

[6] S. Mauw, W. Wiersma, T. A.C. Willemse, Language-driven system design. in HICSS conference, Hawaii, 2002.

[7] A. Cuccuru, C. Mraidha, F. Terrier and S. Gérard, Templatable Metamodels for Semantic Variation Points, in ECMDA conference, June 2007, Haifa, Israel.

[8] OMG, Business Process Modeling Notation Version 1.2, January 2009.

[9] OMG, UML Superstructure V2.1.2, October 2007.

[10] D. Chappell, Introducing Windows Workflow Foundation, 2005, http:/davidchappell.com/.

[11] OMG, Meta Object Facility (MOF) Core Specification Version 2.0, January 2006.

[12] OMG, A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2, 2008.

[13] F. Lagarde, Contribution à la conception de langages de modélisation spécifiques fondés sur UML, PhD thesis, University of Nice, France, 2008.

[14] B. Selic, A Systematic Approach to Domain-Specific Language Design Using UML, in ISORC conference, 2007.

[15] OMG, SPEM version 2.0, April 2008.

[16] OMG, Systems Modeling Language version 1.1, 2008.

[17] OMG, UML Profile for DODAF/MODAF (UPDM), 2008.

[18] OMG, UML Profile for Schedulability, Performance, and Time Specification version 1.1, 2005.

[19] Papyrus UML, http://www.papyrusuml.org/.

[20] OMG, Object Constraint Language 2.0, October 2003.

[21]] Stefan Berner et al, A Classification of Stereotypes for Object-Oriented Modeling Languages, in UML conference, Fort-Collins, United States, 1999.

[22] P. Tessier, S. Gérard, F. Terrier, J.-M. Geib, Using variation propagation for Model-Driven Management of a System Family, in SPLC conference, Rennes, France, 2005.

[23] H. espinoza, An Integrated Model-Driven Framework for Specifying and Analyzing Non-Functional Properties of Real-Time Systems, PhD thesis, University of Evry, France.

[24] H. Espinoza, D. Cancila, S. Gérard, Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems, accepted for publication in ECMDA conference, Berlin, Germany, 2009.

[25] F. Lagarde, F. Terrier, C. André, S. Gérard, Constraints modeling for (profiled) UML models, in ECMDA conference, Haïfa, Israel, 2007.

[26] M. Mernik, J. Heering, A. M. Sloane, When and How to Develop Domain-Specific Languages, ACM Computing Surveys, Vol. 37, No.4, December 2005.