# Leveraging Patterns on Domain Models to Improve UML Profile Definition

François Lagarde[1], Huáscar Espinoza[1], François Terrier[1], Charles André[2], and Sébastien Gérard[1]

[1] CEA, LIST, Gif-sur-Yvette, F-91191, France
{francois.lagarde, huascar.espinoza, francois.terrier, sebastien.gerard}@cea.fr
[2] I3S Laboratory,
BP 121,
06903 Sophia Antipolis Cédex,
France
charles.andre@unice.fr

**Abstract.** Building a reliable UML profile is a difficult activity that requires the use of complex mechanisms -stereotypes and their attributes, OCL enforcement- to define a domain-specific modeling language (DSML). Despite the ever increasing number of profiles being built in many domains, there is a little published literature available to help DSML designers. Without a clear design process, most such profiles are inaccurate and jeopardize subsequent model transformations or model analyses. We believe that a suitable approach to building UML based domain specific languages should include systematic transformation of domain representations into profiles. This article therefore proposes a clearly-defined process geared to helping the designer throughout this design activity. Starting from the conceptual domain model, we identify a set of design patterns for which we detail several profile implementations. We illustrate our approach by creating a simplified profile that depicts elements belonging to a real-time system domain. The prototype tool supporting our approach is also described.

## 1 Introduction

Over the last few decades, domain-specific languages (DSLs) have proven efficient for mastering the complexities of software development projects. The natural adaptation of DSLs to the model-driven technologies has in turn established domain-specific modeling languages (DSMLs) as vital tools for enhancing design productivity.

A widespread approach to the design of a DSML is to make use of the so-called profile mechanisms and to reuse the UML [1] metamodel as the base language. By extending UML elements with stereotypes and their attributes, it is possible to define new concepts to better represent elements of a domain. Despite the ever-increasing number of profiles defined and successfully applied in many applications (Object Management Group (OMG) [2] has adopted thirteen profiles

J. Fiadeiro and P. Inverardi (Eds.): FASE 2008, LNCS 4961, pp. 116–130, 2008.

covering a wide range of modeling domains), building a reliable profile is still an obscure process with very little literature available to help designers.

A study of the currently adopted profiles reveals two existing profile design methods. The first is a one-stage process: the profile is directly created to support concepts belonging to the targeted domain. This method has been adopted for instance by SysML [3] profile designers. The main drawback of such an approach is to narrow down the design space to the implementation level.

A second, more methodical process involves two stages. Stage one is intended to define the conceptual constructs required to cover a specific domain. The product of this stage is usually called the conceptual domain model. In stage two, this artifact is then mapped onto profile constructs. This was the approach used to design UML profiles such as the Schedulability, Performance and Time specification [4] and the QoS and Fault Tolerance specification [5]. Applying this second method allows designers to focus on domain concepts and their relationships before dealing with language implementation issues. The main disadvantage is the time it requires. Finding a correct profile construct to support a conceptual model domain is far from straightforward. Several profile implementations may support a concept, thus obliging the designer to apply cumbersome design heuristics to comply as much as possible with the conceptual model.

We believe that a suitable approach to building UML-based domain-specific languages entails systematic transformation of domain representations (captured in a metamodel) into profiles. For this purpose, we propose a clearly-defined process geared to help the designer. Transformation is based upon a set of design patterns occurring on the domain model for which we provide profile implementations. By doing so, we attempt to improve the accuracy of profiles and facilitate adoption of DSMLs based on UML profiles.

In a previous short-paper [6], we provided arguments in favor of a *staged* process. In the present paper, we give a thorough description of our process and present a set of preliminary results obtained from the prototype tool that we developed.

Most of the results given are based on experience acquired in defining various UML profiles [4, 7]. Among these, the UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [7] required definition of a complex DSML that involved a significant collaborative effort by domain experts.

This paper is organized as follows: Section 2 explains the reasons for devising a DSML and advocates use of profiling mechanisms. Section 3 details the identified key stages of our process and progressively introduces a set of profile design guidelines. Section 4 presents the prototype. Section 5 describes related work and Section 6 gives our conclusions.

## 2 Why (How and When) to Create a DSML

Most of the reasons for creating DSLs are also valid for DSMLs. The expected benefits have been described in previous studies [8, 9]. The most widely shared of these advantages are that DSMLs:

- allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain,
- embody domain knowledge and thus enable the conservation and reuse of this knowledge.

The easiest approach to design a DSML is to reuse an existing metamodel. A common approach is to develop libraries, also called domain-specific embedded languages (DSELs) [10], which are easy to integrate into almost all frameworks. In some cases, constraints may be expressed, to confine the use of an existing language to a specific domain. While both these options are applicable to almost all languages, model-driven technologies afford two novel approaches which are usually classified according to their mechanisms. They are:

**heavyweight extension:** this approach allows designers to extend and modify the source language as required to create a new DSML. In a recent paper [11], Alanen and Porres provide a comprehensive overview of available mechanisms, and outline the possibilities of subset and union properties in formal definition of new metamodels. This extension method is particularly suited to cases where there are few identifiable relationships to existing languages,

**lightweight extension:** this approach is restricted to the use and extension of an existing Meta-Object Facility (MOF) [12] based metamodel which cannot be modified. It is supported with the standard profiling mechanism.

Development of a DSML may lead to heavy investments, and recourse to such tool requires strategic decisions. Clearly, there are no systematic criteria for such decision making. A balanced rule would be to avoid creating a DSML wherever possible, by instead learning to better identify the design needs and recognize them in existing languages.

Unfortunately this rule cannot always be applied; and regardless of the mechanisms chosen, the following difficulties remain:

- designing and maintaining a DSML is a time-consuming task,
- it is hard to strike a suitable balance between domain-specific and general purpose language constructs,
- the costs of educating DSML users could be high.

Among the approaches available, standard mechanisms can be regarded as the most sustainable solutions. Such mechanisms benefit from the low cost of support tools and the limited investments involved in the learning process. We consequently advocate creation of metamodels with profiling mechanisms, since the latter are standard built-in mechanisms and present a more steeply downward-sloping learning curve for multiple and long term projects.

## 3 Design Activities Flow

This section discusses the four phases proposed for our process. Fig. 1 shows the process workflow with its different outputs, from conceptual domain definition to profile creation:

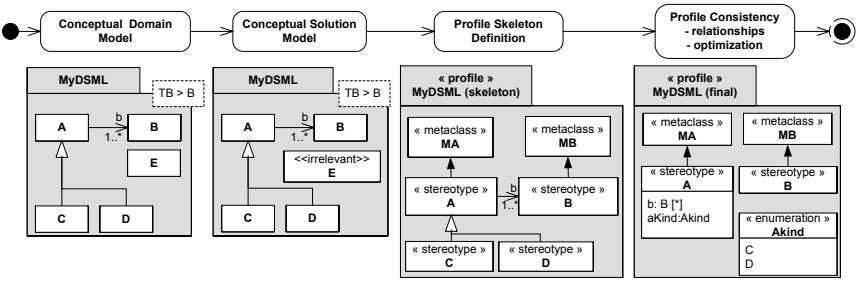


**Fig. 1.** Main workflow

1. The designer gives a coarse-grain description of the space problem using a UML class diagram. This first iteration is the conceptual domain model.
2. This model is then tailored to a conceptual solution model,
3. Solution entities are transformed into potential stereotypes and extensions to UML metaclasses are given. This operation results in a profile skeleton,
4. The profile skeleton is used as an analysis artifact. At this stage, profile designers decide how domain representations are to be supported in the final profile.

### 3.1 Conceptual Domain Model

Domain model definition is the initial phase of our process. It identifies the required concepts and their relationships in order to describe a problem area, also called problem space. This design activity is independent from technological solutions and thus lets the designer focus on domain constructs.

Much attention has been paid on domain analysis and many existing studies [13, 14] describe the techniques used to support this process. One of the central issues is how to manage the common concepts and variable concepts of a problem area to enable reuse assets for different projects.

One way of incorporating such flexibility into a modeling process is to make use of template mechanisms. These allow formal identification of parametric information that can subsequently be refined. Recourse to templates in the design work flow then makes it necessary to determine at what stage they can be included. While templates are frequently used for system modeling, few studies [15,16] have examined their uses for metamodeling. We believe that to better meet the *design for reuse* criterion, we need to capture variability at the start of our process. We therefore introduce this facility at the conceptual domain definition stage.

We use the UML Class package to build the conceptual domain model. The advantages are that DSML designers are familiar with its concepts (e.g., inheritance, associations, and packages) and that the UML Class package has a support for template mechanisms.

To tangibly illustrate this discussion, we have created a simplified DSML for the real time domain, called Simple Real Time System, inspired by the profiles

SPT and MARTE. The SRTS package in the left panel of Fig. 2 is an excerpt of this conceptual model. A SchedulableResource depicts any resources to be scheduled and maintains a reference to exactly one Scheduler. The latter is characterized by its SchedulingPolicy. Schedule and SchedulingJob are entities for modeling a queue of jobs waiting to be executed.

The model embodies two parametric concepts shown in the dashed box in the upper right corner of Fig. 2. The reasons for this choice are that an hypothetical list of schedule policies or schedulable resources would have hampered reusability of the conceptual domain model.

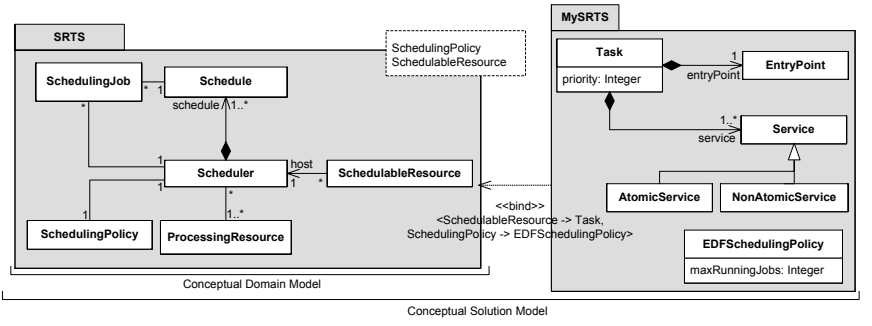


**Fig. 2.** Conceptual domain/solution model

### 3.2 Reducing the Problem Space: Building a Conceptual Solution Model

Some concepts may be required within the conceptual domain model to enhance the readability and completeness of a problem area description; these same concepts may not, however, be appropriate for solving a given problem. The subsequent stage in design consists of identifying the concepts which will serve as solution assets and to create bindings to the templates.

In order to indicate that a concept is merely present for description purposes, with respect to the targeted domain application, we introduce an *irrelevant concept* supported through a stereotype named Irrelevant.

The left and right panel in Fig. 2 together constitute the result of this stage (details have been omitted to preserve readability). It is then decided to identify ProcessingResource, SchedulingJob and Schedule as irrelevant concepts since their roles are limited to making the problem more understandable.

Templates bindings have been declared. The parameter SchedulableResource is substituted for Task. This concept is made up of one EntryPoint and a set of Services. The Service concept is further specialized into PremptibleService and NonPremptibleService. Scheduling policy is bound to the EDFSchedulingPolicy.

### 3.3 Profile Skeleton Definition

This phase initiates creation of the profile. Each conceptual solution entity is transformed into a stereotype; and all the relationships, such as generalizations

and associations, are maintained. Note that the capability of stereotypes for participating in associations has only recently been included in version 2.1 of UML. Exceptions are the irrelevant concepts, since they are not intended for use in practical modeling situations.

At this point, the main design decision activity is mapping stereotypes to the UML metaclasses. A set of extension associations must therefore be manually established from the stereotypes to the UML metaclasses.

Fig. 3 is the profile skeleton. We have specialized the metaclass Class to support both Scheduler and Task, while considering that Operation metaclass is better suited to supporting EntryPoint and Service. EDFSchedulingPolicy reuses the DataType concept because it is a primitive type of our DSML.

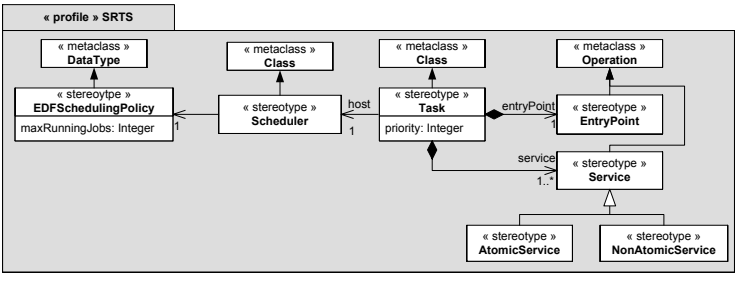


**Fig. 3.** Profile skeleton

### 3.4 Profile Consistency Analysis and Profile Optimization

To ensure definition of a well-formedness profile, a designer should ensure that none of the newly introduced concepts must conflict with the *semantics* of the extended metamodel. The term semantics must be interpreted in a broad sense. It namely encompasses the informal class description or class semantics of the UML metamodel, along with the structural definition of the metamodel (e.g., there is no possibility of creating additional meta-associations or meta-attributes). That is why, in [1, p.650], the reference metamodel is deemed to be *read-only*.

If this vital rule is usually satisfied in simple metamodel design, its strict application to a broader metamodel definition is difficult. The complexity of the UML metamodel, when combined with the (voluntary) ambiguities of UML semantics, means that there can be no guarantee of success other than the trust placed in the designer's skills.

Although a formal mechanism to assess well-formedness of profile and consistency with the reference metamodel seems hard to establish, in-depth exploration of the metamodel may help the designer select the most suitable profile implementation. The following section presents a set of formal guidelines enabling profile creation based on design patterns on the profile skeleton. We have divided these guidelines into two parts. First we identify all the patterns related to association/composition in the conceptual domain for which the profile solution is given, and, second, we provide detailed rules for profile optimization.

**Dealing with Meta-association**

It is common to create new meta-associations between concepts which must in some way be reflected in the profile.

The designer is then responsible for finding the implementation that best represents the meta-association. This mainly entails mapping onto two profile constructs; either a stereotype attribute is defined to maintain a reference to the associated concept or a submetaclass of a Relationship is made mandatory at level $M1$. Occasionally, the relationship may already be sufficiently represented in the metamodel and an OCL rule is enough to reinforce this intent. Even if these solutions express comparable information, they result in different profile applications scenarios that may affect its usability.

In our example, Scheduler has an association link with EDFSchedulingPolicy. The designer might choose the first solution, and the stereotype supporting the concept of Scheduler in turn embeds an attribute; or he/she might use a subclass of a DirectedRelationship such as Dependency to better represent this association.

Since associations at level $M2$ may be represented differently at level $M1$, some means of distinction should be provided. However the lack of information available to instantiate an association across more than one level makes the designer accountable for her/his decisions. This inability to carry information has already been the focus of research efforts. In [17], the authors refer to it as *shallow instantiation*. Their initial observation is that information concerning instantiation mechanisms is bound to one level. A recent study [18] formulates a similar diagnosis and its authors propose to recognize *induced associations* in models.

To provide a means for selecting the proper profile implementation, we identify different association patterns between concepts. Firstly, we detail the modeling case where the specialized metaclasses supporting the two associated concepts have an association. Secondly, we elaborate on the case in which no association is found. We then explore the constructs that make it possible to distinguish between a composite association and a non composite association. Fig. 4 summarizes the associations of interest here, along with the identified solutions.

**Existing meta-association.** In our example, Task is associated with EntryPoint and both extend the metaclasses Class and Operation. The first precaution is to make sure that this association does not violate the structure of the extended metamodel. Since there is at least one composite association from Class to Operation, the two introduced concepts satisfy this requirement. This meta-association may be mapped in three ways: attribute creation, directed relationship support or OCL enforcement.

We formally identify pattern recognition as follows:

**Identification 1 (meta-association identification).** *Let A and B be two stereotypes extending the metaclasses MA and MB respectively. If A has one (composite) association with B with member end rb (lower bound multiplicity n, upper bound multiplicity m) and if in MA there is at least one (composite) association with MB, then A and B are part of a meta-association pattern.*

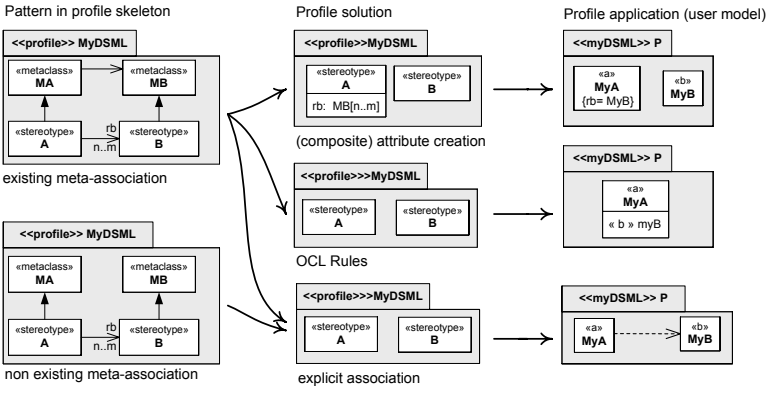


**Fig. 4.** Association patterns and profile solutions

An intuitive mapping solution is to declare an attribute in a stereotype to maintain information with the associated concept. In order to faithfully represent the information conveyed by the association: the type of the property must be that of the base class of the associated stereotype, and OCL rule enforcement must ensure correct application of stereotypes to the associated elements. We have generalized these considerations via the following profile solution:

**Solution 1.1 (attribute creation).** The property rb is used to create an attribute in A:

- the attribute rb is created with a multiplicity equal to the multiplicity of rb,
- attribute type is MB,
- stereotype ≪A≫ becomes a context for an OCL rule ensuring that values of rb are stereotyped ≪b≫:

```
context A inv:
 self.rb->forAll(e:MB | not(e.extension_B.oclIsUndefined()))
```

Use of a relationship such as Dependency or Abstraction provides a flexible means for modeling references between DSML elements. A typical example of this is the allocation concept of SysML. The advantage of this approach is that relationships are readily recognizable, since they are represented with an explicit element.

In order to ensure that elements stereotyped ≪a≫ have DirectedRelationship links with enough elements stereotyped ≪b≫ we need to create an OCL constraint in the context of the stereotype ≪A≫. Because the base class supporting A is not affected by the DirectedRelationship, navigating to linked elements is difficult and involves exploring the namespace of the element stereotyped ≪A≫. We have only given a fragment of the OCL rule ensuring the lower bound multiplicity of property rb.

**Solution 1.2 (explicit association).** OCL constraints must be created to make sure that are enough DirectedRelationship links with element stereotyped ≪b≫ to comply with the multiplicity of rb.

```
context A inv:
self.base_MA.namespace.ownedMember->select(e: NamedElement |
 (e.oclIsTypeOf(uml::DirectedRelationship)) and
 (not(e.oclAsTypeOf(uml::DirectedRelationship))
        .target.extension_B.oclIsUndefined()))->size()>=n
```

The above two solutions either create a stereotype attribute or make explicit use of a subclass relationship metaclass. Alternatively we can consider that the association is sufficiently represented in the metamodel. In that case, we must ensure stereotypes application.

For instance, based on the association from Task to EntryPoint, we may decide that a profile application is correct as long as a Class stereotyped ≪task≫ owns enough Operation stereotyped ≪entryPoint≫.

The following OCL expression gives the generic profile solution (as illustration, only the lower bound multiplicity is verified). Note that this solution narrows down the scope of the associated elements to the owned members of MA.

**Solution 1.3 (OCL rules).** A becomes a context for an OCL declaration in charge of ensuring compliance with the multiplicity constraint obtained from property rb.

```
context A inv:
self.base_MA.ownedElement->select(e: Element |
  (e.oclIsTypeOf(MB)) and
  (not(e.extension_B.oclIsUndefined())))->size()>=n
```

**Non-existing meta-association.** This subsection considers failure of the previous meta-association pattern identification, i.e stereotypes having associations that are not in the metamodel. The profile requirement that none of the introduced elements conflicts with the semantics (in this case the structure) of the extended metamodel is then not met.

Strict application of this requirement leads to the conclusion that there is a modeling *error* and to rejection of the offending association in the profile. The solution is to identify another metaclass to support one of the two stereotypes. For example, the designer may look at the parents of one of the base metaclasses, which are higher concepts with fewer constraints (the metaclass Element can be virtually used to support any extensions: it can also be part of any association with any other element, since Element has a cyclic composition with Element).

However, recourse to another metaclass may affect the semantics of the concept and not be a faithful support. To strike a balance between flexibility and adherence to the guideline, we suggest solutions that do not affect the base metaclass. Among the formerly identified patterns, the explicit use of a DirectedRelationship meets this requirement.

**Handling composite concepts.** Thus far, we have formulated no hypothesis about the kind of aggregation involved in the associations in the profile skeleton. However, this characteristic plays a key role in domain modeling and requires a reliable profile support.

The semantics of a composite association, also known as an *aggregation* or *containment relationship* have a common meaning in traditional programming language. They indicate that the aggregated object's lifecycle is controlled by the aggregating object. In models, composition is used to denote hierarchy and ownership. Composition between metaelements results in a deletion constraint between instances of the metaelements. In profiles, this definition has no straightforward interpretation. Stereotypes are statically applied to model elements and their application cannot therefore be coerced.

We have identified two possible solutions to support this design intent. The first, confines the use of composite association to concepts that extend metaclasses already having a composite relationship. In this situation, the attribute pattern solution and OCL solution comply with the composition constraint. Nevertheless, this approach raises much the same issues as already mentioned for the OCL solution and implies that composite elements are owned members of the aggregating element. By doing so, deletion of the base class supporting the aggregating concept results in deletion of the aggregated concepts.

To overcome this limitation, we opt for the attribute pattern solution and the aggregation kind meta-attribute of the stereotype attribute. This solution involves a support tool (discussed in section 4) to ensure correct deletion. When a base class supporting a stereotype is deleted, their attributes are inspected, if the kind of aggregation is composite then the referenced model elements are deleted.

**Solution 1.4 (composite attribute creation).** The same scheme is used as for the attribute creation solution. Additionally, composition information is reflected in the meta-attribute aggregation kind.

### Optimization

The next identified subtask in profile creation is profile optimization for the purpose of minimizing the number of stereotypes. This is intended to preclude a proliferation of stereotypes and resulting applications that may be detrimental to the readability and understandability of models.

We propose two reduction patterns, as illustrated in Fig. 5. The first of these entails hiding a concept in an already existing UML concept with a stereotype attribute, and the second subsumes domain concepts in enumeration data types.

**Hiding an existing concept.** In our example, the Task concept is made up of EntryPoints. One alternative is to keep both concepts as stereotypes. However, we might consider that the EntryPoint entity, which is considered equivalent to a behavioral feature concept, is sufficiently represented by UML operation elements, and thus save one stereotype declaration. Based on this reasoning, we

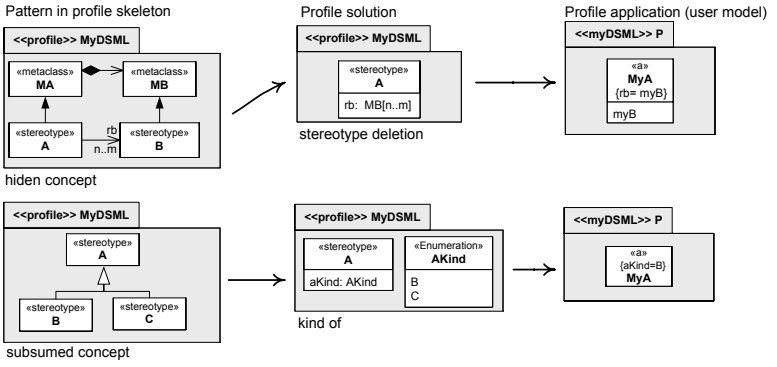


**Fig. 5.** Optimization patterns

can create a ≪Task≫ stereotype and use the association end named entryPoint to create an attribute typed as UML Operation. Obviously, this pattern only succeeds if the aggregated stereotype does not embed an attribute or participate in another association or generalization.

**Identification 2 (hidden concept).** *Let A and B be stereotypes complying with conditions expressed in the meta-association pattern. If A does not have additional stereotype attributes or additional stereotype association/generalization relationships, then B can be hidden in the target profile.*

**Solution 2 (stereotype deletion).** The attribute solution is reused with the difference that stereotype ≪B≫ is no longer required.

**Subsuming a concept.** This pattern occurs in inheritance relationships when the specialized concepts are used to describe taxonomies and do not carry any information (domain attributes or associations) except the concept itself.

In our example, Service is further specialized into AtomicService and NonAtomicService. The profile skeleton assigns one stereotype to each. As a reduction strategy, we might wish to replace the specialized concepts by an enumeration named ServiceKind, with a set of literals named NonAtomicService and AtomicService (see final profile in 6). Service thus contains an attribute named serviceKind, to indicate which kind of service we are referring to.

**Identification 3 (subsumed concept).** *If a set of stereotypes (e.g., B and C) specializes another stereotype A, and if the former do not embed any property or participate in any association or generalization relationship, then the substereotypes could potentially be reduced.*

**Solution 3 (kind of).** An enumeration AKind is created with literals matching the reduced substereotypes (e.g., C and D). An attribute in A allows reference to the enumeration data type: AKind.

**Final Profile**

6 is one final profile resulting from consecutive application of the identified transformation patterns to the initial profile skeleton. It embodies a set of OCL rules associated with the selected patterns. The derived profile clearly outlines the difference between the conceptual domain model and its implementation. Every association has been translated either into a stereotype attribute or into proper OCL constraints. The immediate result is that the profile can no longer be considered the main support for conveying how concepts are related to each other. However, this side effect has no impact on understanding of the DSML since we consider the domain model as the main artifact for declaring concepts.

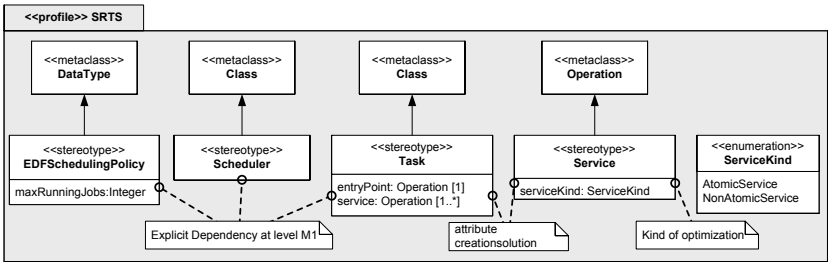


**Fig. 6.** One final profile

## 4 Tool Prototype and Evaluations

### 4.1 Tool Prototype

Our methodology involves several model manipulations which may be error-prone if they are performed manually. This is especially true for writing OCL enforcements and exploring metaclasses relationships.

Since we have identified profile design patterns for which we then identified pattern solutions, it is possible to partially automate our proposed approach. We have evaluated its viability by developing an Eclipse plug-in for the Papyrus UML tool as a means for partially automating the process. The plug-in's user interface is a sequence of dialogues corresponding to each of the identified key steps.

The main difficulty arises from the several profile implementation possibilities for an identified pattern. The designer is responsible for selecting the most appropriated solution and the transformation must take place interactively.

To provide a flexible support for describing rules we have defined a metamodel (not presented here due to limited space). Elements of this metamodel describe concepts for running diagnoses on models. This is done by a protocol definition that can execute tests on the model. Each such test associates a recognition concept, which is a context for declaring OCL helpers in charge of detecting a pattern in the model (like code smells for models), with the corresponding

solutions. A solution is broken down into elementary fix. If a test succeeds, then one or more solutions are suggested to the designer with an order of preference.

Once a pattern is identified and a solution has been selected, the following operation is to process it with the relevant fix. We used Epsilon Object Language (EOL) [19] as an imperative OCL language for model manipulations. The Fix becomes a place to declare theses expressions.

Fig. 7 shows the dialog box that indicates which solutions may be applied to the Task. For each of these solutions, basic model transformations are also given.

As part of this prototype, we also defined mechanisms to handle the composite attribute creation solution. When a model element supporting a stereotype is deleted, if a composite attribute exists, a dialog box appears to confirm deletion of the composed elements.

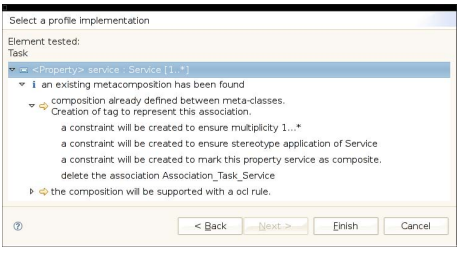


**Fig. 7.** Dialog box for selecting a profile implementation

### 4.2 Evaluation and Feedback

This plug-in has made it possible to evaluate our approach on the MARTE profile. We have considered the final adopted MARTE profile (realtime/07-05-02) as the profile skeleton. The presented patterns/solutions have been described with a model that conforms to the protocol metamodel. In addition, new tests have been defined to take into account additional rules (e.g., naming convention, optimization of stereotype extensions). This led us to define more than 15 tests. The accuracy of the resulting profile is enhanced by the OCL constraints declared in each stereotype and some modeling design mistakes (stereotypes extending a metaclass and one of its sub-metaclass e.g. Class and Classifier) were identified.

## 5 Related Work

As already stated earlier, very little published material is available on design of domain-specific UML profiles.

In [20], Fuentes and Vallecillo pointed to the need for first defining a domain metamodel (using UML itself as the language) to clearly define the domain of the problem. In more recent work [21], Bran Selic has described a similar staged development of UML profiles and gives useful guidelines for mapping domain constructs to UML. The initial version of the SPEM [22] profile, presents general guidelines for transforming a metamodel into a profile. Our proposal also leverages use of a conceptual model but attempts to go a step further by identifying patterns on the conceptual model as a means for inferring a reliable profile.

Concerning conceptual modeling, Gerti Kappel et al described in [23] a process for “lifting metamodels into ontology models”. A metamodel is considered here

as an implementation-specific artifact that makes concepts hard to understand. It identifies a collection of patterns that help to transform metamodels into equivalent ontology constructs. Our research entails the opposite approach, i.e. transforming conceptual domain models into well-formedness profiles.

A precedent for our type of approach was established by the AUTOSAR (AUTomotive Open System ARchitecture) project, whose modeling framework defines an interesting mechanism for building templateable metamodels. This entails a special "language", defined by the UML profile for Templates. This language identifies a set of common patterns occurring in (meta)modeling (e.g., types, prototypes, instances). In our approach, we attempt to define a more systematic and flexible approach to designing the conceptual model and its implementation.

## 6 Conclusion

This paper presents a systematic approach to the design of UML profiles by leveraging use of the conceptual domain model. For this purpose, we have elaborated on a staged process that helps the designer throughout the profile design process. Starting from the conceptual domain we, determine a set of regularly occurring design pattern for which we identify profile solutions in terms of stereotypes as well as OCL constraints.

Our approach is illustrated by a running example for which we define concepts for depicting a simple real-time system domain. These domain concepts are transformed step-by-step into an equivalent profile.

To evaluate the viability of our approach, we present the Eclipse plug-in developed for this purpose. This plugin is a promising development that appears to have other potential applications. It could be used whenever a model transformation requires intervention from the designer to select a rule transformation from among several possibilities.

We are currently completing our plug-in to handle traceability requirements. This would allow designers to easily navigate between the different representations of an element: from conceptual domain to profile and vice versa. It would also allow storage of designer's decisions and permit profile regeneration.

## References

1. Object Management Group: Unified Modeling Language, Superstructure Version 2.1.1 formal/-02-03 (2007)
2. Object Management Group: OMG, `http://www.omg.org`
3. Object Management Group: Systems Modeling Language (SysML), Specification, Adopted version, ptc/06-05-04 (2007)
4. Object Management Group: UML Profile for Schedulability, Performance and Time (SPT) formal/2005-01-02
5. Object Management Group: UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and mechanisms formal/06-05-02

6. Lagarde, F., Espinoza, H., Terrier, F., Gérard, S.: Improving UML Profile Design Practices by Leveraging Conceptual Domain Models. In: Automated Software Engineering (November 2007) (short paper)
7. Object Management Group: UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) 1.0 finalization underway
8. van Deursen, A.v., Klint, P., Visser, J.: Domain-specific languages: an annotated bibliography. SIGPLAN Not. 35(6), 26–36 (2000)
9. Consel, C., Marlet, R.: Architecturing software using a methodology for language development. In: Symposium on Programming Language Implementation and Logic Programming, vol. 1490, pp. 170–194 (1998)
10. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. 37(4), 316–344 (2005)
11. Alanen, M., Porres, I.: A metamodeling language supporting subset and union properties. Software and Systems Modeling (June 2007)
12. Object Management Group: Meta-Object Facility formal/2006/01-01
13. Frakes, W.B., Kang, K.: Software reuse research: Status and future. Software Engineering, IEEE Transactions on 31(7), 529–536 (2005)
14. Czarnecki, K.: Overview of Generative Software Development. Unconventional Programming Paradigms, 313–328 (2004)
15. Cuccuru, A., Mraidha, C., Terrier, F., Gérard, S.: Enhancing UML Extensions with Operational Semantics - Behaviored Profiles with Templates. In: Model Driven Engineering Languages and Systems (November 2007)
16. Emerson, M., Sztipanovits, J.: Techniques for Metamodel Composition. In: OOPSLA, 6th Workshop on Domain Specific Modeling, pp. 123–139 (2006)
17. Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. In: Software and Systems Modeling (2007)
18. Burgués, X., Franch, X., Ribó, J.: Improving the accuracy of UML metamodel extensions by introducing induced associations. Software and Systems Modeling (July 2007)
19. Kolovos, D., Paige, R., Polack, F.: The epsilon object language (eol), 128–142 (2006)
20. Fuentes-Fernández, L., Vallecillo-Moreno, A.: An Introduction to UML Profiles. UML and Model Engineering V(2) (April 2004)
21. Selic, B.: A Systematic Approach to Domain-Specific Language Design Using UML. In: International Symposium on Object and Component-Oriented Real-Time Distributed Computing, vol. 00, pp. 2–9 (2007)
22. Object Management Group: Software Process Engineering Metamodel (SPEM) formal/2005-01-06
23. Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M.: Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. In: Model Driven Engineering Languages and Systems, pp. 528–542 (2006)