A Metamodel for SDL-2000 in the Context of Metamodelling ULF

Joachim Fischer, Michael Piefel, and Markus Scheidgen

Humboldt Universität zu Berlin
Institut für Informatik
Unter den Linden 6
10099 Berlin, Germany
`(fischer|piefel|scheidge)@informatik.hu-berlin.de`

Abstract. Today the syntax of many languages is defined by using context-free grammars. These syntax definitions suffer from a major drawback: grammars do not allow the definition of abstract, reusable concept definitions. Especially in families of related languages, where multiple languages often share the same concepts, this limitation leads to unnecessary reproduction of concept definitions and a missing shared base for these related languages.
Metamodels can contain inheritance hierarchies of concepts; thus multiple specifications can reuse and refine existing shared concept definitions. Therefore we propose a method to develop metamodels from existing syntax definitions. We explain our method by applying it to SDL-2000. The method starts with a mapping from BNF grammars into simple preliminary metamodels. Then, by supplying a relation between elements of these simple metamodels and abstract concepts, these metamodels are automatically transformed into metamodels that use existing descriptions of abstract concepts and thus allow a shared basis of common abstract concepts definitions.

1 Introduction

In the ongoing research on model driven software engineering the relations between different modelling languages are a key point. The approach in [1] uses model transformation between *eODL*[2] and *SDL-2000*[3] to drive software projects from design to implementation. Such a technology requires language alignment. To build such relations as transformation rules, between these languages, we needed unified specifications for both of the participating languages. The need of unifying the SDL-2000 grammar based syntax definition with the eODL metamodel started our research on developing a metamodel for the SDL-2000 syntax. But the metamodel of SDL-2000 only attacks the tip of a far more common problem.

ITU-T recommends a long series of formalized languages, such as MSC, ASN.1, TTCN and the already mentioned eODL and SDL. These languages were developed independently using different specification techniques. Unfortunately, this resulted in languages that are hard to relate to each other. But language alignment is crucial for model driven engineering. It is important to know where and how these languages can be used together, how to profit from an integrated use of these languages. Consequently, the ITU-T started the *Language Coordination Project*[4] with the goal to unify the mentioned languages, thus to form the *Unified Language Family (ULF)*. The *Language Coordination Project* considers two methodologies to achieve a coordinated syntax definition: a common syntax for BNF grammars (a common meta-metagrammar) and metamodelling.

We figured the metamodelling technology to be more appropriate to define the abstract syntax of a language, its concepts and its structure, than to use context free grammars. We give the reasons for this opinion in the following comparison of BNF grammars and metamodels:

Context free grammars, used in language specifications, are mostly in BNF. The BNF syntax was developed to specify concrete language syntax. It is a mathematicly exact method to determine which words are in the described language and which are not. To achive this, grammars use rules; these rules specify a set of productions; these productions represent a set of words, words that form the language described by the grammar. However, what grammars do not provide are means for rule refinement or generalization, they do not allow modularization. It is not possible to refine rules to form generalization hierarchies or to build logical structures by using a namespace mechanism of some sort. Structure and abstraction hierarchies are always flat.

This is a severe grammar disadvantage. The following two profits, known from object-oriented languages, cannot be taken from a grammar based specification. First, generalization, and thus refinement and reuse, as well as logical structures, provided through mechanisms like namespaces and packages, are vital concepts for compact and easy to understand language specifications. With these features not only the words of a language can be specified, but the internal structure of the described language can be defined. This allows a natural evolution of language specifications and easy tool development. Second, in the context of language families, a package of shared abstract concepts, concepts that are refined in different languages, can directly align the concepts of different languages. This allows the notation of inter language relations at specification time, and thus allows unification of the participating languages.

These two points make clear that a modern language specification must offer more than a pure syntax definition, it must show the language's internal structure and make it accessable to other language specifications. This is to allow the notation of relations between different languages. Therefore we believe that a syntax specification technique must provide the following two important meta-meta-concepts. First, generalization; the building blocks of a syntax definition must be refinable. One must be able to define abstract concepts and one must be able to use them in multiple concrete refinements in the definition of the same and in the definition of different languages. Second, structural composition; to use the same concepts in different language definitions and to allow bigger language definitions, namespaces for structured specification must be provided.

Both concepts, generalization and structural composition, are provided by modern metamodelling architectures; Atkinson's work[5] gives a good introduction into the subject. As object-oriented modelling platforms, metamodelling architectures provide a generalization mechanism as well as a namespace mechanism. Metamodels are designed for abstract syntax, or better static concept definition. As a side effect the meta-language as well as the languages to be modelled are object oriented modelling languages. Due to that common nature, an expert in the specified language is often also well trained in the used specification technique.

After we were aware of the potential of metamodelling for language specification, we faced the problem of providing this metamodelling features to already existing, grammar based language specifications. Therefore we propose a method that allows to develop a metamodel from an existing grammar based syntax definition. The method fulfils two requirements: It is partially automated to avoid the error-prone task of manual grammar to metamodel transformation and the resulting metamodels use a set of abstract concept definitions. These abstract concepts are to be identified by a language expert and can be shared in the specification of different languages or in the specification of a hole family of languages.

To prove and explain our method, we applied it to a part of the abstract SDL-2000[3] syntax. The developed metamodel covers all of SDL's structural concepts. The reason for modelling only structural concepts is that structural concepts are by far the best researched concepts, and various abstract concepts for many languages are already identified and modelled[6]. Thus we can fully concentrate on the method and general metamodelling aspects. But of course the method is independent from what is modelled; therefore it also works for other parts of language specifications like the description of behaviour or data concepts.

We tried to stay as general as possible, to make the method applicable with most existing metamodelling architectures. But a few requirements had to be made. The metamodelling architecture has to consist of at least four layers[7]. The method is made for a strict[5] metamodelling environment; but that does not imply that it will not work for a loose metamodelling architecture as well. The used M_3 layer model must include a generalizable class concept, associations, namespaces and a package concept.

The mentioned properties are fulfilled by the most important metamodelling architectures: MOF 1.4[8], UML 2.0[6], OML[9], and even formal metamodelling techniques like VPM[10]. We used the meta-metamodel in [8] and implemented our method by using JMI[11], which is a MOF 1.4 based metamodelling standard[8].

In section 2 we give an overview over the developed method. The sections 3 and 4 cover the necessary steps of that method. The concluding section 5 compares the resulting SDL-2000 metamodel with the original grammar based syntax definition, it summarizes the paper, and it gives some further research perspectives.

2 A method for metamodelling existing languages

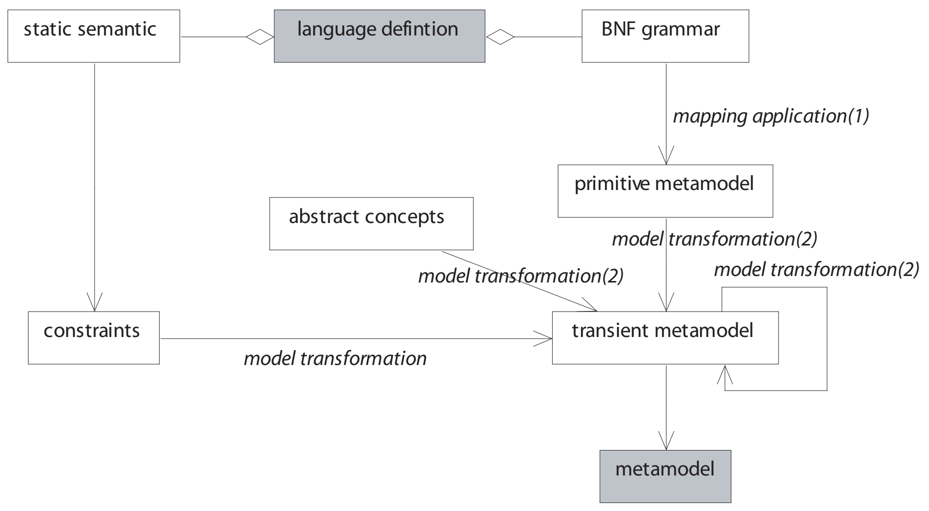

Fig. 1. The steps involved in the presented metamodel development.

The steps involved in our method are presented in figure 1. The whole process is based on an already existing syntax and static semantics specification. This paper addresses only the grammar part of language definitions; it does not cover aspects of static semantics. A complete presentation of the method, including static semantics concerns, can be found in [12].

The development of a metamodel from a BNF grammar includes two steps. First a preliminary metamodel is generated from the existing BNF grammar. We required our method to do this step automatically, because a manual transformation from a grammar to a metamodel would be cause to many errors due to human failure. Metamodels are more expressive than grammars. Thus it was easy to find a grammar to metamodel mapping. Section 3 describes this mapping.

Of course, implied by this automatic transformation, the resulting metamodel only uses meta-concepts already provided by BNF grammars. Thus neither generalization nor structural composition are used. All concept definitions reside in the same namespace and they form a flat hierarchy.

To use the more advanced concepts generalization and structural composition some human input is needed. Along our second method requirement two things have to be provided by a language expert. First, abstract concepts must be identified and modelled. Of course there are different levels of abstraction. Some concepts are very abstract and shared by many languages, like generalization, namespaces or a type system, others are more specific to the modelled language like SDL's instance sets. In section 4.3 we show the models of a package consisting of general abstract concepts and a package consisting of the abstract SDL concepts, that we could identify. The second kind of information that must be provided by a human expert is: Which concrete language concept is a refinement of which identified abstract concept? We call this information a semantic mapping. A model transformation engine can now use both inputs, abstract concept definitions and semantic mapping, to automatically convert the preliminary primitive metamodel into the final metamodel. The semantic mapping and the transformation engine are explained in 4.3.

3 Generating metamodels from BNF grammars

BNF grammars, primarily used in the definition of computer languages, are basically context free grammars. The rules of such grammars replace a non-terminal with a regular expression containing non-terminals and terminals. These regular expressions may use the composition, alternative, arbitrary, at-least-one and optional constructs. An example of such rules, taken from the abstract procedure definition syntax of SDL-2000, is presented in figure 2.

```
Procedure_definition :: Procedure_name
    Procedure_formal_parameter*
    [ Result ]
    Procedure_graph;

Procedure_name = Name;

Procedure_formal_parameter = In_parameter
    | Inout_parameter
    | Out_parameter;
```

Fig. 2. SDL-2000 abstract syntax example.

For a better understanding of the concepts used in BNF grammars and their relationships figure 3 presents a meta-metamodel for BNF grammars. Now we are modifying this grammar meta-metamodel until it becomes a meta-metamodel for common metamodelling architectures. Along the modifications made on the meta-meta-level we show a mapping from BNF concepts to the concepts of metamodelling. Thus creating transformation rules for the meta-level, which are used in the automatic grammar to metamodel conversion.

The first modification is to use concepts out of the concept space of object-oriented metamodelling, as long these concepts can replace the respective grammar concepts isomorphicly. Figure 4 shows a more general model than 3. It uses the metamodelling concepts class and relation for the grammar concepts symbol, terminal, non-terminal and rule. Unfortunately, the concept *Expression* and its specializations have no equivalent in metamodelling.

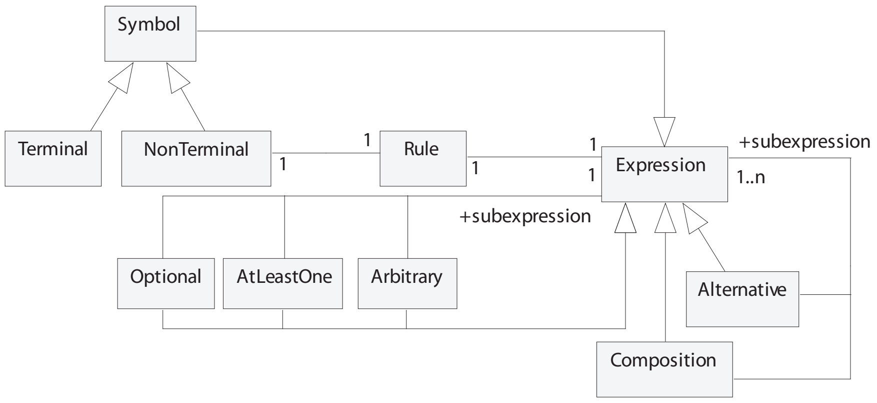

Fig. 3. BNF grammar meta-metamodel.

When you think about the semantics of *Composition* and *Alternative*, they turn out to be those of *Association* and *Generalization*. Associations allow the linear composition of classes, alias symbols, and specialized classes are alternative realizations of the general class, alias symbol. This leads to the idea of mapping compositions and alternatives to associations and generalizations. The expression kinds that describe multiplicities, can be compared to the multiplicity of association ends in object-oriented metamodelling. A modified meta-metamodel realizing this ideas is shown in figure 5.

Expressions allow the recursive composition of an unlimited depth of subexpressions. To respect that, *Class* in figure 5 is modelled as a generalization

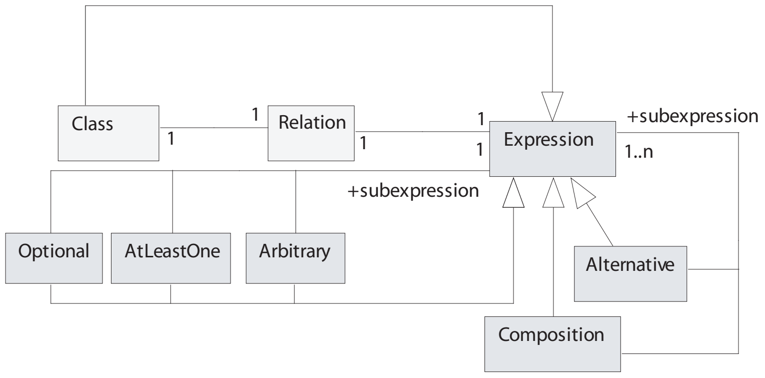

Fig. 4. Homomorphous modifications.

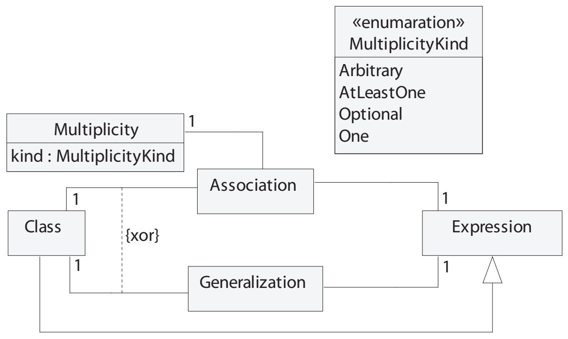

Fig. 5. Associations and generalizations for relations rather than compositions and alternativs as expressions.

to *Expression*. Thus allowing *Expression* to relate with itself. But there is no reason not to completly identify *Class* with *Expression*, see figure 6.

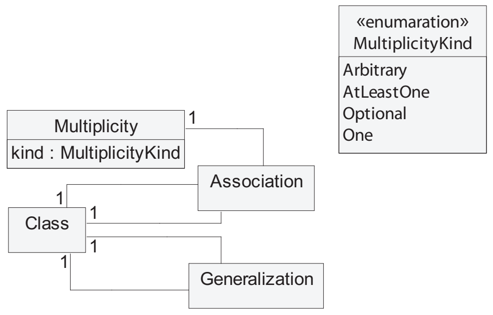

Fig. 6. A more usual meta-metamodel.

Now it is easy to build transformation rules along the meta-metamodel evolution from figure 3 to figure 6. The only problematic thing are the names for classes that represent sub-expressions. In BNF grammars sub-expressions are nameless and are only separated by the use of parentheses. But classes in meta-modelling are named model elements. We simply name the classes that represent sub-expression rather than symbols with unused new names.

The developed BNF grammar to metamodel transformation rules are:

1. Every symbol is represented through a class.
2. A rule with a single symbol on the right is represented through an association that associates the class representing the left hand symbol with the class representing the right hand symbol.

3. A rule with an composition on the right is represented through an association for every composed sub-expression.
4. A rule with an alternative on the right is represented through a generalization for every alternated sub-expression.
5. A sub-expression consisting only of a single symbol is represented through that symbol's class.
6. A sub-expression that is a composition or an alternative is represented through a new class, with a so-far unused name. The composition or alternative is transformed as in 2 or 3, but with the new class as the left hand representative.
7. A sub-expression of multiplicity kind, that is part of a composition, is transformed to an equivalent multiplicity kind of the proper association end.
8. An expression of multiplicity kind or a sub-expression of multiplicity kind, that is part of an alternative, is represented through a new class with a so far unused name. An association is introduced between that new class and the class that represents the multiplied sub-expression, with proper multiplicity.

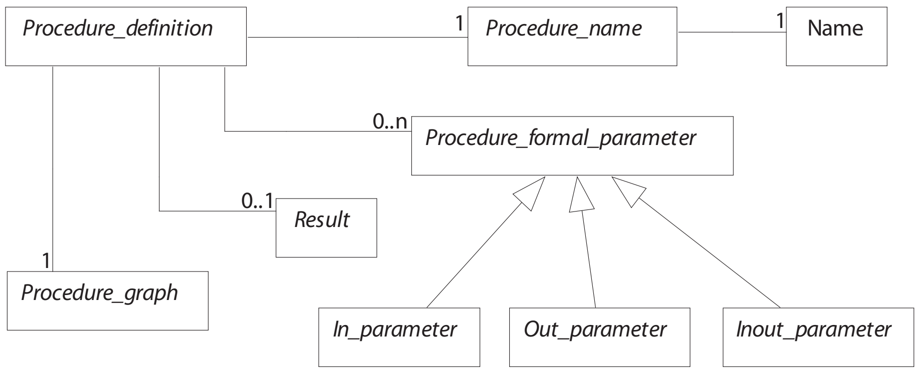

Fig. 7. The example grammar part mapped into a metamodel.

We implemented a simple tool, consisting of a BNF grammar parser, the implemented transformation rules, and a XML generator, producing the metamodel in XMI format. XMI is a standard format for metamodel exchange. This tool called *agramm* successfully transforms the abstract SDL-2000 grammar. Figure 7 shows a part of it. It shows that syntax part that is shown in the grammar example in figure 2 at the beginning of this section.

4 The use of abstract concepts

4.1 Beyond the grammar to metamodel mapping

The metamodels one get by applying the described mapping from a BNF grammar can be called primitive at best. The reason is, of course, that those primitive metamodels can only be as expressive as the original grammar is. We identified three categories of problems in that the primitive metamodels must be improved.

First, those primitive metamodels suffer from the same drawback as the original grammars: They do not use and reuse abstract concept descriptions. One example are the structural type concepts in SDL-2000, namely *Agent type definitions* and *Composite-state type definitions*. These two concepts are generalizable, they can be instantiated, they are namespaces for a number of other SDL concepts, and so forth. But these abstract concepts are specified separately for both of that elements, instead of being defined once and then reused. Especially being a namespace is a property that even more SDL concepts like procedures and packages share. Furthermore, namespace is an abstract concept used by many other languages; take eODL for an example.

The second problem is that grammars are very limited in their meta-meta-concepts, and so there are many metamodelling features that are not used by the automatic generated metamodels. In the grammar example in figure 2, both *Procedure_definition* and *Procedure_name* are described by a symbol. Therefore both concepts are modelled by a class in the generated metamodel. Of course, that is bad metamodelling technique. The name of a procedure should rather be modelled through a string attribute. The same problem lays in the modelling of associations. Metamodelling concepts like navigability, aggregation, etc. cannot be stated in grammars, and so they are not used in the generated metamodel, even if their usage would be appropriate.

The third problem is what can be called textual syntax rudiments. These are concepts like *identifier* and *qualifier*. In a text based language they are needed to identify objects. To do so they represent a logical relation between the object definition and its usage, like the definition of a variable and its usage in an expression. In a metamodel and its model instances these concepts are not necessary. These relations can be modelled by associations and their instances, called links. In other words these are concepts that already exist as meta-meta-concepts and have not to be redefined. Of course, *identifier* and *qualifier* are needed in concrete model notations to represent those relations, but they serve no purpose in an abstract language definition.

Of course the first problem category and its solution, that is the introduction of abstract concept definitions, is the most interesting. The usage of abstract concepts leads to better structured and reusable syntax definitions and thus presents the biggest advantage in comparison with a grammar based syntax definition.

In addition to that, it turned out that, in the case of SDL-2000, the most concepts that cause problems of the second and third category are potentially abstract concepts. Potential means that these are concrete concepts that should be replaced by abstract definitions. Thus these faulty concrete realizations will vanish when abstract concepts are introduced. For example: The name feature in the procedure example causes a problem of category two; the concept name is realised through a class instead a string attribute. But it also is a potentially abstract feature. There are a lot of SDL concepts that have a name property.

That is why it is most likely that after the introduction of an abstract named model element concept, the concrete concept *procedure definition* has lost its distinctive name property and inherits it from the abstract concept, instead. The same can be said about *identification* and *qualification*. They are often used by concepts that are potentially abstract, and thus they will not be used after well modelled abstract concepts have been introduced. In this paper we concentrate on the more interesting category one: The introduction of abstract concept definitions.

4.2 Abstract concept definitions

First we present the abstract concept definitions that we used for the SDL-2000 metamodel. There are different levels of abstractions. Some concepts are so general that they are used in virtually every object-oriented language, others are more specific and may only be reused among related languages or only within one language.

Figure 8 shows the abstract structure concepts used by most object-oriented languages. A detailed explanation and documented development of that model can be found in [12]. The resemblance to the most known languages might strike the reader's eye. They are also used in the successful metamodels of UML and the meta-metamodel of MOF.

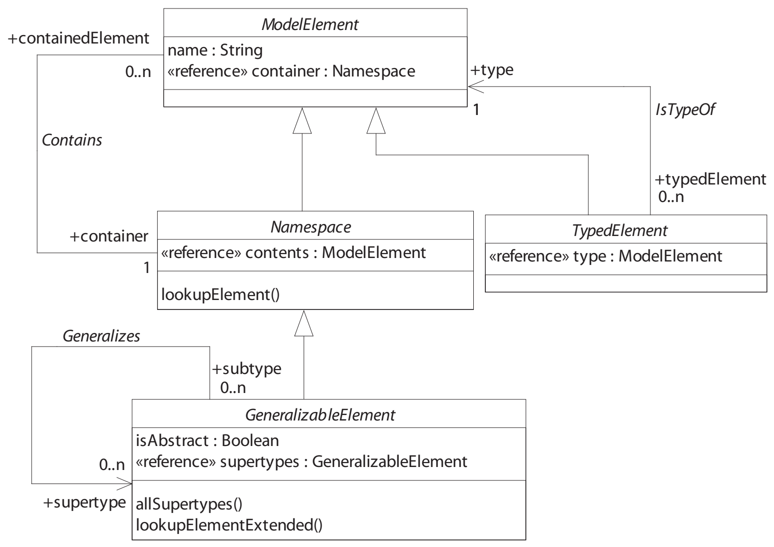

Fig. 8. Abstract concepts.

But even more abstract concepts could be obtained from SDL's syntax itself. Even if they may turn out to be more specific, perhaps distinctive to SDL, they

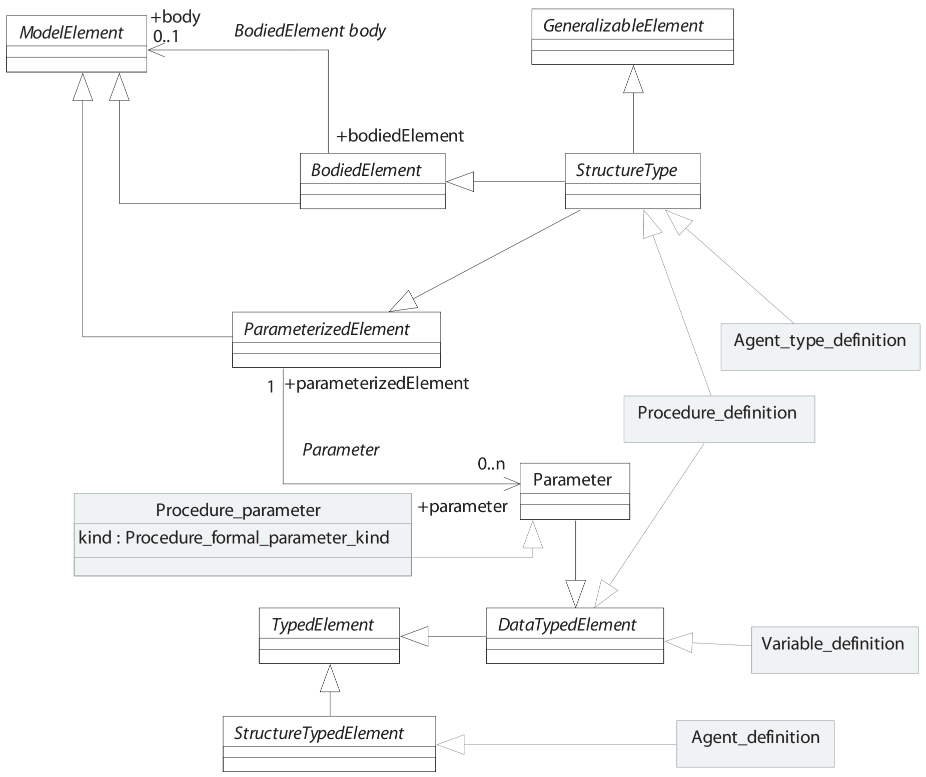

Fig. 9. Abstract SDL-2000 concepts.

still allow a more compact and therefore easier to understand and easier to use metamodel. Figure 9 presents the additional abstract concepts that we were able to identify in the SDL-2000 syntax. A few concrete concepts, those that are marked grey, are shown too. That is to ease the understanding and shorten the necessary explanations. A few remarks:

– Many SDL concepts reference a *body* of some sort. Procedures for example must contain a state automaton defining their behaviour. This state automaton is referred as a *body*. The same is true for process typed *Agents* or the bodies in *Composite-state types*. To respect the varying nature of bodies, they are modelled to be the most abstract concept: *ModelElement*.
– Parameters are used by a variety of SDL concepts. *Agent types, Procedures, Composite-state types* have parameters. Even if *Procedure* uses a special form of parameter, the parameter itself is a typed element in any case.
– In SDL-2000 two type concepts coexist. A type is something that describes a set of instances or values. In SDL a type can on the one hand be a data

type, like a *Signal definition* or a primitive data type and on the other hand a type can be a structure type like *Agent type* or *Composite-state type*.
– Structure types are instancable, generalizable, parameterized types. Therefore structure types are a combination of the generalizable concept, parameterized concept and the body possession concept.

As a reminder: one may criticize that the displayed model allows unwanted instances, that procedure for example may contain a non-procedure parameter, or a structure-typed element may reference a data type. Obviously some restrictions have to be added to the model. Actually these constraints are considered static semantics and are not covered by this paper, but [12] addresses that matter by using a predicate logic formalism to further limit the set of possible metamodel instances.

4.3 Combining primitive metamodels with abstract concept definitions

Now we have sets of abstract metamodel elements and a generated primitive grammar-based metamodel. But how to combine these model elements to form a single metamodel? Two things must be realized: First, the concrete concepts must be marked as specializations of the introduced abstract concepts. And second, features and rudiments of concrete elements that are already defined or realized by the corresponding abstract model element must be removed. To accomplish this task, we use model transformation.

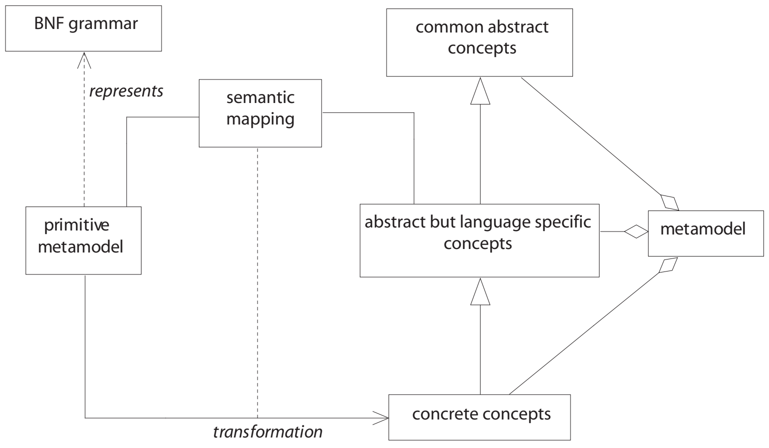

Fig. 10. Transforming the primitive metamodel.

Figure 10 shows the basic idea of this transformation. We already have the grammar that is represented by the primitive metamodel, and we already have common as well as more language specific abstract model elements. To complete the metamodel, we have to transform the concrete concepts of the primitive metamodel to actually become specializations of the abstract model elements.

To do so some information from a language expert is needed. That is information about the nature of the concrete language concepts, something that can be given through a *semantic mapping* between the primitive metamodel elements and the abstract model elements. This mapping must say which concrete concept is a specialization of what abstract concept and how it refines the abstract concept. With this information the transformation itself can be done automatically.

How it works: The semantic mapping is a partial relation that assigns concrete metamodel elements to the most appropriate abstract model element. The mapping of model elements does not only involve a mapping between classes, but a mapping between relations as well.

```
agentTypeDefinition = new SdlStructureTypeAdaptor(                      1
          "Agent_type_definition");                                     2
agentTypeDefinition.addSupertypeType("Agent_type_defintion");           3
agentTypeDefinition.addBodyType("State_machine_definition");            4
agentTypeDefinition.addParameterType("Agent_formal_parameter");         5
agentTypeDefinition.addContainedType("Agent_type_definition");          6
agentTypeDefinition.addContainedType("Procedure_definition");           7
agentTypeDefinition.addContainedType("Agent_definition");               8
     ...                                                                9
```

Fig. 11. An example taken from the semantic mapping used for the SDL-2000 metamodel.

As an example figure 11 presents a part of the semantic mapping used for the SDL-2000 metamodel. The first line assigns *Agent_type_definition* to be a specialization of the abstract concept class *StructureType*. The second line maps agent type definition's association with itself to be a specialization of the generalization association introduced by one of agent type definition's new super meta-classes: *GeneralizableElement*. Line three maps agent type definition's association with state machine definition to be a specialization of the body association introduced by another new super type of agent type definition: *BodiedElement*. Line four refines the inherited features of the abstract concept *ParameterizedElement*, the other lines refine the inherited features of the abstract concept *Namespace*.

This mapping is actually Java code. That is because we realized the transformation using *JMI*[11] a Java based API for metadata management. For every abstract concept an adaptor class was written in Java. *SdlStructureTypeAdaptor* is such an adaptor. The inheritance hierarchy of the adaptors is aligned to the hierarchy form by the corresponding abstract concepts. Thus the super types of *SdlStructureTypeAdaptor* are *GeneralizableElementAdaptor*, *Parameter-*

izedElementAdaptor, *BodiedElementAdaptor*, *NamespaceAdator* and *ModelElementAdator*.

For every abstract concept class a adaptor class was written, for every concrete class a adaptor instance is created by the semantic mapping. The constructor that is used maps the concrete concept class to the abstract concept class. This means the constructor uses JMI to introduce a new generalization relation between the abstract concept class and the concrete concept class that is provided through the constructor's parameter.

For every abstract association or attribute a method was written. The method is owned by the adaptor class for the abstract concept class, that the association or attribute is originated in. For every concrete association a method call is used. This Java method call maps the concrete association to the corresponding abstract association. Therefore the Java method deletes the old concrete association, originally generated through the grammar to metamodel generator, and replaces it with a refinement of the abstract association. This refinement is done through the addition of a constraint. The concrete association is identified by taking the concrete class for one end and taking the class provided through the Java method's parameter for the other end.

For example, look at line 2 of the previous semantic mapping example 11. Originated in SDL's abstract grammar, a concrete association between *Agent_type_definition* and *Agent_type_definition* exists in the primitive metamodel. This association refers to the inheritance relationship between two agent types. Line 2 maps the abstract association *Generalizes* of *StructureType*'s and therefore *Agent_type_definition*'s meta- superclass *GeneralizableElement*. The invoked Java method removes the original concrete association and replaces it by a constraint that restricts the abstract inherited *Generalizes* association to allow only links between two instances of *Agent_type_definition*. The mapping of *StructureType* associations then continues for the abstract associations *Contains* and *ElementBody*.

This way the mapping works as a chain of commands that transforms the metamodel according to the semantics given by the mapping. After the whole mapping is applied, all concrete concepts are specializations of abstract ones, all concrete associations and rudiments have been removed and replaced by constraints that restrict abstract associations or attributes. The only things left are a few concrete concepts for which no appropriate abstract concepts could yet be identified. For these concepts some manual transformations have to be made.

In particular, the explained SDL-2000 metamodel lacks an abstract concept for the SDL communication concepts *channel* and *gate*, as well as the concrete agent-instance- set concept that uses minimum and maximum instance numbers. An abstract relation concept may be very useful, because relations occur in many languages as well as in the behaviour of SDL itself. Therefore it should be introduced when the metamodel family grows, and the abstract basis becomes larger. The second left-out concept on instance sets is, as far the authors know, unique to SDL, and therefore only a concrete description is needed.

5 Conclusions

We presented a method that allows the development of metamodels from existing syntax definitions. The presented method shows the following characteristics:

- It is partially automatic and thus less error-prone.
- The only human input that is needed is a model of abstract concepts; a mapping from concrete, grammar originated, model elements to abstract model elements; and a transformation rule for every abstract concept, that transforms generated concrete elements to become specializations of abstract elements.
- Modelled abstract elements and transformation rules are reusable and extendable and can be used in the development of multiple metamodels. That is an even bigger advantage in the modelling of language families.

We applied and tested our method on SDL-2000. The resulting metamodel shows the following characteristics and advantages, when compared to the original grammar based syntax definition.

- Due to the extensive usage of abstract model elements and refinement the resulting SDL metamodel is compact and easy to understand. Abstractions are already noted in the metamodel itself, they have not to be explained in additional text. The inheritance hierarchy of concepts can be directly and naturally used in the development of object-oriented SDL tools.
- The metamodel includes and is based on a reusable refinable abstract basis. This basis can easily be reused in metamodels of languages that share the same concepts. A shared set of abstract concepts can be used in a unified specification of language families and allows a direct alignment of languages that share the same abstract concepts.
- The concepts of the metamodel are well structured by the use of packages. Due to such structural concepts like namespaces and packages, metamodels can be easily combined and related to each other.

These advantages over grammars are mostly based on the metamodelling concepts generalizability and namespaces. Both are concepts that grammars can not support.

But even if we see many advantages of the metamodelling side, we have to admit that metamodels cannot replace grammars in the specification of concrete syntax. When it comes to the task of defining and parsing textual notation the formal foundations and exact semantics of grammars cannot be yet replaced. But in defining the concepts of a language in an abstract manner to derive semantics definitions, tool development and human understanding from it, the advantages of metamodels are superior.

We developed a series of tools to support the method application and metamodel development. A tool called *agramm* was created that allows automatic transformation from BNF grammar to MOF metamodels. The API *mmm*, based on *JMI*[11], is a framework for metamodel transformations that uses refinable transformation rules.

There are a few problems that should be addressed by further research. First, the metamodelling of behaviour concepts is not yet satisfactory. Abstract behaviour concepts must be identified and should be used for the completion of the SDL-2000 metamodel. Second, with the metamodel for SDL-2000 we started to build an abstract set of concepts. It should be used and extended in the metamodelling of other ULF languages. Third, the relations of abstract ULF concepts to the concepts used in UML should be researched for easier UML profiling or other language integration. And fourth and last, the most challenging problem is metamodelling of dynamics. The formal specification of SDL-2000 showed a way to specify a language's dynamic semantics, using *Abstract State Maschines*[13]. Integration of that knowledge into metamodelling architectures could be result in an unified metamodelling technique for reusable syntax and semantics definition.

References

1. Böhme, H., Fischer, J.: eODL and SDL in combination for components. In: Fourth SDL and MSC Workshop. (2004)
2. ITU-T Z.130: Extended Object Definition Language (eODL), International Telecommunication Union (2003)
3. ITU-T Z.100: Specification and Description Language (SDL), International Telecommunication Union (2002)
4. Reed, R.: Language Coordination Project – Revised information – Workshop results, International Telecommunication Union (2003) TD 3145.
5. Atkinson, C.: Meta-Modeling for Distributed Object Environments. In: 1st International Enterprise Distributed Object Computing Conference. (1997)
6. UML 2.0: Infrastructure Final Adopted Specifcation. Object Management Group (2003) ptc/2003-09-15.
7. Crawley, S., Davis, S., Indulska, J., McBride, S., K.Raymond: Meta-meta is better-better! In: IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems(DAIS'97). (1997)
8. MOF 1.4: Meta Object Facility, Version 1.4. Object Management Group (2003) formal/2002-04-03.
9. Handerson-Sellers, B., Firesmith, D., Graham, I., Page-Jones, M.: OPEN Modeling Language (OML) Meta-model Specification, Version 1.0. (1996)
10. Varró, D., Pataricza, A.: VPM: Mathematics of Metamodeling is Metamodeling Mathematics. Journal of Software and Systems Modelling (2003) 1–24
11. JMI: The Java Metadata Interface(JMI) Specification(Final Release). Java Community Process (2002) JSR-000040.
12. Scheidgen, M.: Metamodelle für Sprachen mit formaler Syntaxdefinition, am Beispiel von SDL-2000. Humboldt Universität zu Berlin (2004) master thesis.
13. Gurevich, Y.: Evolving Algebras 1993: Lipari Guide. In Börger, E., ed.: Specification and Validation Methods. Oxford University Press (1995) 9–36