A Model-Driven Approach for Telecommunications Network Services Definition

Vanea Chiprianov[1,3], Yvon Kermarrec[1,3], and Patrick D. Alff[2]

[1] Institut Telecom, Telecom Bretagne, UMR CNRS 3192 Lab-STICC
Technopole Brest Iroise, CS 83818 29238 Brest Cedes 3, France
{vanea.chiprianov,yvon.kermarrec}@telecom-bretagne.eu
[2] BT-North America
2160 E. Grand Ave, El Segundo, CA 90245, United States
[3] Universite europenne de Bretagne, France

Abstract. Present day Telecommunications market imposes a short concept-to-market time for service providers. To reduce it, we propose a computer-aided, model-driven, service-specific tool, with support for collaborative work and for checking properties on models. We started by defining a prototype of the Meta-model (MM) of the service domain. Using this prototype, we defined a simple graphical modeling language specific for service designers. We are currently enlarging the MM of the domain using model transformations from Network Abstractions Layers (NALs). In the future, we will investigate approaches to ensure the support for collaborative work and for checking properties on models.

1 Introduction

Present day Telecommunications customer-centric market, with its high demand rate and fierce competition, imposes a fast pace to service providers. To shorten the concept-to-market time, the service providers need to make their service definition more efficient by designing the product right-the-first-time. The current service definition process is largely based on trays of documents being exchanged between service designers and programmers. We contend that capturing the service definition knowledge into a computer-aided, model-driven (Sect. 2) design tool shortens this process and enables capitalization on previous experience.

Capturing domain specific knowledge may be done by iteratively constructing a Domain Definition Meta-model (DDMM), as presented in Sect. 3 and represented by the central entity in Fig. 1 (in this figure we represent with filled ellipses what we have already done; in parentheses we indicate the toolkit we used). Starting from the DDMM, we can define one or several Domain Specific Languages (DSLs) (Sect. 4) which increase the performance of service designers. The approach of defining several DSLs and semi-automatically generating tools (editors, checkers) for them from an information model has been previously proposed for policy-based management systems [1].

Defining a telecom service is a collaborative work which involves several designers. They need specific support for team work and especially a solution to

M. Oliver and S. Sallent (Eds.): EUNICE 2009, LNCS 5733, pp. 199–207, 2009.

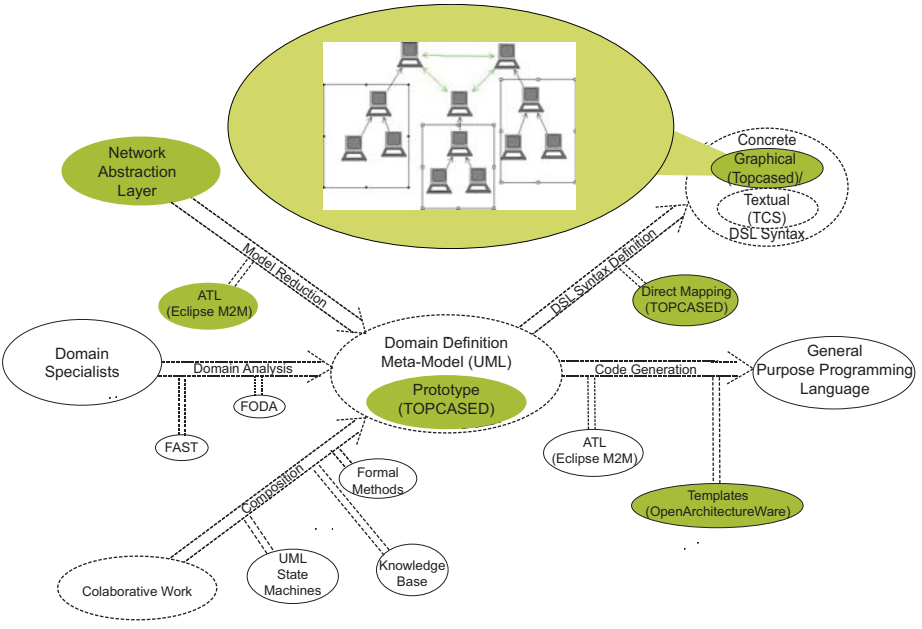

Fig. 1. Telecommunications-specific Modeling

put together their individual work into a composed definition of the service. We discuss this aspect in Sect. 5, together with the need to verify and validate the definition of a service. The DDMM can support the service designers in their collaborative work when defining a new service and can also be used for verifying properties on models that were defined using the aforementioned DSLs. Therefore, as highlighted by [2] also, the DDMM is central to our approach.

2 Model Driven Engineering

Model Driven Engineering (MDE) is a software engineering approach concerned with bridging the conceptual gap between the problem and implementation domains by using as primary artifacts of development abstract models that describe the system from multiple viewpoints and by providing automated support for analyzing models and transforming them to concrete implementations. The basic principle of MDE is "*everything is a model*" [3].

"*A model represents reality for the given purpose; the model is an* ***abstraction*** *of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a* ***simplified*** *manner, avoiding the complexity, danger and irreversibility of reality*" [4].

MDE main challenges have been recently surveyed by [5]:

1. Modeling language: how can one create and use problem-level abstractions in modeling languages;
2. Separation of concerns: how can one use multiple, possibly overlapping models of the same system described from several viewpoints, possibly in heterogeneous languages;
3. Model manipulation and management challenges: define, analyze and use model transformations, maintain traceability links among model elements, maintain consistency among viewpoints and use models during runtime.

In our approach of modeling services, we meet the same challenges. To address them, we propose:

1. A DSML, presented in Sect. 4, defined using a model-based approach.
2. An approach based on collaborative work, as presented in Sect. 5.
3. Using model transformations extensively, starting from constructing the DDMM from an NAL, as presented in Sect. 3, to connecting to tools that use formalisms that enable checking properties on models 5.

3 Elaborating the Domain Definition Meta-Model

We approached the definition of the DDMM with an iterative method in mind. We started with the definition of a simple MM, for prototype purposes. This prototype is aimed at defining a simple virtual private network (see Fig. 4). The prototype consists of a *Network*, which may contain several inner networks and several *Node*s. The nodes are either *Computer*s, *Internet* or *Router*s; they are connected by links which constitute outlinks for the source nodes, and inlinks for the target nodes. The routers can be either customer edge routers (*CE*) or provider edge routers (*PE*). Each PE and CE has an *Interface*, which contains a virtual routing and forwarding (*VRF*) table containing the VrfRouteTargets and information about the neighboring PEs (*BgpIpv4AddressFamilyNeighbor*s). PEs use the Border Gateway Protocol (*BgpRoutingProtocol*) to communicate with each other. We also enriched the DDMM with validation rules [6], thus enabling domain level validation. As tool we chose TOPCASED [7], a strongly model oriented system engineering toolkit for critical and embedded applications.

We are currently working on enlarging the DDMM. For this, we start from existing NALs which are specified in UML and simplify them to suit the needs of service designers (see top left ellipses in Fig. 1). We specify the reduction rules using the ATL [8] model transformation language. Consequently, the reduction rules are written as model transformation rules. As presented in Fig. 2, a), model transformations are models themselves, conforming to their MM. They take as input one or several source models and produce as output one or several target models. In our case, Fig. 2, b) the transformation is endogenous (i.e.; the MM of the input model(s) is the same as the MM of the output model(s) - UML) and has one source (i.e.; NAL) and one target model (i.e.; DDMM). We exemplify the transformation rules in listing 1, which presents a rule that copies to the output model all classes from the input model that have at least one method.

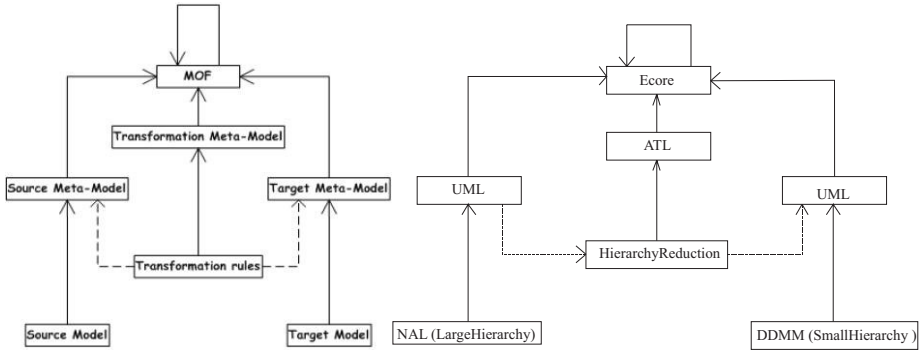

Fig. 2. a) MM-based model transformation (from [3]) and b) DDMM construction by model transformation

```
module HierarchyReductionUML; -- Module Template
create SmallHierarchyUML : UML from LargeHierarchyUML :
    UML;

rule Class {
    from
        cs : UML!Class(
            cs.ownedOperation->notEmpty())
    to
        ct : UML!Class(
            name <- cs.name,
            package <- cs.package,
            ownedOperation <- operationLst
            ),
            operationLst : distinct UML!Operation foreach
                (oper in cs.ownedOperation.asSequence())(
                    name <- oper.name)
}
```

listing 1

We chose this approach because a NAL already captures a big part of the service domain, but in a much more detailed manner than necessary for a designer. By eliminating all entities that are unknown to service designers and shrinking the inheritance hierarchies, we believe we can elaborate a MM that is close to the service domain. In addition, such a MM would have the advantage of being easy to map to existing NALs. More details about this approach of constructing the DDMM by model transformation can be found in [9]. In the future, we consider enriching the DDMM obtained by reduction from NALs iteratively, by

using specific domain analysis methods, such as Family-oriented Abstractions Specification and Translation [10] or Organization Domain Modeling v2 [11].

4 A Simple Graphical Telecommunications Specific Modeling Language (SGTSML)

"*A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions,* ***expressive power*** *focused on, and usually restricted to, a* ***particular problem domain***."[12] The DSL development methodology has been extensively presented by [13].

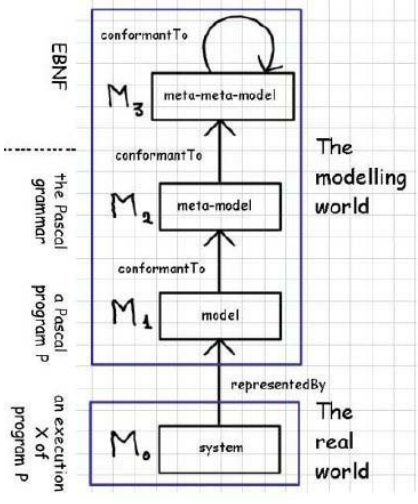

Fig. 3. Modelware and Grammarware (from [3])

On one hand, being a programming language, the definition process of a DSL has to be consistent with the definition process of any programming language. Consequently, an abstract syntax, a concrete syntax and a semantics have to be defined. On the other hand, [14] consider that "*a DSL is a set of coordinated models*". The two points of view are not at all contradictory, as illustrated by Fig. 3, in which the correspondence between modelware and gramarware is illustrated (i.e.; to the language to describe the grammar of a programming language, in modelware it corresponds the meta-meta-model; to the grammar of the programming language it corresponds the MM and to the program itself it corresponds the model). The abstract syntax of a DSL can be modeled as the DDMM (Fig. 4), the concrete syntax can be represented as a "display surface" MM and the semantics can be obtained through a transformation model between the DDMM and, either the MM of a DSL with a precise execution, or the MM of a general purpose programming language (e.g.; Smalltalk).

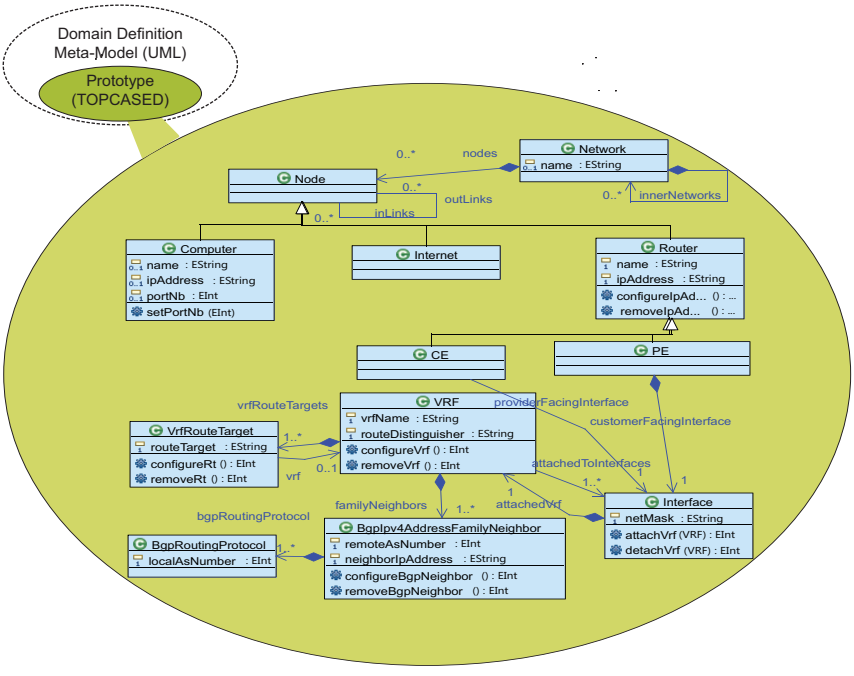

Fig. 4. Abstract Syntax of SGTSML

We tackled the construction of the DDMM in Sect. 3. For the concrete syntax (see top right filled ellipses in Fig. 1) we consider that a graphical syntax will be much easier to use by service designers, as it provides a synthetic, high-level view of the system being considered. Therefore, we defined one using TOPCASED, which has a feature that allows automatic generation of graphical editors for DSLs based on their MM. To describe the semantics of our SGTSML we decided to use the semantics of an existing general purpose programming language, Smalltalk (see right ellipses in Fig. 1). Consequently, we defined templates for code generation towards Smalltalk, using OpenArchitectureWare [15]. More details about the definition of SGTSML can be found in [6]. In the future, we intend to extend the concrete graphical syntax to represent the entire enlarged DDMM and to define a concrete textual syntax too, using tools such as TCS [16], or more classical approaches, such as compilers or translators.

5 Towards Collaborative Work and Checking Properties on Models

Using the modeling language, the designers will define a service. However, because a service is a complex entity, several designers are required to collaborate for its definition. Therefore, we must provide them with adequate communication

and interaction tools. We plan to construct a knowledge base which will contain the decisions that are taken during service definition and their justifications, a design rationale system [17]. We are considering also a form of behavior modeling and model composition, but have yet to investigate and decide between UML State Machines, formal methods, ontologies or other approaches.

We are studying as well methods to ensure the models produced by service designers are valid. We envisage defining model transformations from the MM (abstract syntax) of the modeling language towards the MM of formalisms that are capable of verifying a number of properties of interest on the models defined using the modeling language. However, we have yet to identify the properties of interest and the best formalisms to check them.

6 Advantages and Disadvantages of Taking a Model-Driven Approach for Service Definition

Rapid tool prototyping. When defining a language, tools like editors, syntax checkers, compilers need to be provided together with the language. Evolving these tools together with the language can be a very time consuming task when using the classic language theory. To reduce this time, MDE proposes meta-tools: tools to define other language-specific tools. With their help, language tools can be rapidly defined and maintained during the evolution of the language. Moreover, the freed resources can be redirectioned towards other activities.

Independence from the implementation platform. The service model build using our DSML will not contain any platform details, it will be a platform independent platform (PIM). This ensures the reusability of the model, which contains only the business logic. Of course, this model needs to be implemented on several platforms, so the details of each platform are described in a platform specific model (PSM). Consequently, the PIM should contain concepts which are abstractions of those from the PSMs. This is what we are trying to ensure by building our DDMM (i.e.; the MM of the PIM) from existing NALs (i.e.; PSMs).

Iterating definition of the DDMM. The construction of the DDMM is a difficult task. It consists in extracting the abstractions of the domain, verifying that the MM is complete (i.e.; all necessary concepts and actions can be expressed), taking care that it remains in the intended scope. This is a long process, which takes years and implies many changes. Moreover, the domain itself evolves over the years. Having the flexibility of iteratively defining the DDMM ensures that the tools and the models will keep the pace with the changes in the domain.

Tool connection through interchangeable models. MDE proposes an XML representation of the models. Like in the case of web services, the models become interchangeable between different tools, programming languages. Connecting to existing model-based schedulability, performance or other property analysis tools becomes possible.

Poor meta-tool configuration power. In order to provide its most important feature - domain specificity - a DSL needs a specific concrete syntax. For this, the language meta-tools need to be highly configurable (e.g.; for a graphical syntax, allow to define the position of the pallet). This is not true for the current tools (e.g.; the icon definition problem with TOPCASED described in [6]).

7 Conclusions

Our purpose is to replace the current paper-based service design process with a more computer-aided version. For this, we prototyped a DDMM and used it to define a SGTSML. We are currently working at enlarging this DDMM through simplifications from existing NALs. Using a model-driven approach has provided us with powerful advantages. In the future, we plan to integrate support for collaborative work and for checking properties on models in the language, but have yet to investigate the best approaches.

References

1. Barrett, K., Davy, S., Strassner, J., Jennings, B., van der Meer, S., Donnelly, W.: A Model Based Approach for Policy Tool Generation and Policy Analysis. In: Proceedings of the IEEE Global Information Infrastructure Symposium (2007)
2. Fahy, C., Davy, S., Boudjemil, Z., van der Meer, S., Loyola, J., Serrat, J., Strassner, J., Berl, A., de Meer, H., Macedo, D.: Towards an Information Model That Supports Service-Aware, Self-managing Virtual Resources. In: Proc. of the 3rd IEEE internat. workshop on Modelling Autonomic Communications Environments (2008)
3. Bezivin, J.: In search of a basic principle for model driven engineering. Novatica Journal 2, 21–24 (2004)
4. Rothenberg, J.: The nature of modeling. In: Artificial Intelligence, Simulation, and Modeling (1989)
5. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: FOSE 2007, pp. 37–54 (2007)
6. Chiprianov, V., Kermarrec, Y.: Model-based DSL Frameworks: A Simple Graphical Telecommunications Specific Modeling Language. In: Actes des 5 émes journées sur l'Ingénierie Dirigée par les Modéles, Nancy (2009)
7. Farail, P., Gaufillet, P., Canals, A., Le Camus, C., Sciamma, D., Michel, P., Cregut, X., Pantel, M.: The TOPCASED project: a Toolkit in Open source for Critical Aeronautic Systems Design. In: ERTS (2006)
8. Bezivin, J., Dupe, G., Jouault, F., Pitette, G., Rougui, J.: First experiments with the ATL model transformation language: Transforming XSLT into XQuery. In: 2nd OOPSLA Workshop on Generative Techniques in the context of MDA (2003)
9. Chiprianov, V., Kermarrec, Y.: An Approach for Constructing a Domain Definition Metamodel with ATL. In: Model Transformation with ATL, First International Workshop, Nantes (to be published, 2009)
10. Coplien, J., Hoffman, D., Weiss, D.: Commonality and Variability in Software Engineering. IEEE Softw 15, 37–45 (1998)
11. Simos, M., Anthony, J.: Weaving the Model Web: A Multi-Modeling Approach to Concepts and Features in Domain Engineering. In: ICSR 1998 (1998)

12. Deursen, A.V., Klint, P., Visser, J.: Domain-specific languages: an annotated bibliography. SIGPLAN Not. 35, 26–36 (2000)
13. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. 37, 316–344 (2005)
14. Kurtev, I., Bezivin, J., Jouault, F., Valduriez, P.: Model-based DSL frameworks. In: OOPSLA 2006, pp. 602–616 (2006)
15. Features, C.: openArchitectureWare 4.2. Technical report, Eclipse (2007)
16. Jouault, F., Bezivin, J., Kurtev, I.: TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In: Proceedings of the 5th internat. conf. on Generative programming and component engineering (2006)
17. Regli, W.C., Hu, X., Atwood, M., Sun, W.: A survey of design rationale systems: Approaches, representation, capture and retrieval. Eng. with Computers (2000)