Adding Abstraction and Reuse to a Network Modelling Tool using the Reuseware Composition Framework

Jendrik Johannes[1] and Miguel A. Fernández[2]

[1] Technische Universität Dresden, Nöthnitzer Str. 46, 01187 Dresden, Germany
jendrik.johannes@tu-dresden.de
[2] Department of Broadband Service Platforms, Telefónica R&D, Valladolid, Spain
mafg@tid.es

Abstract. Domain-specific modelling (DSM) environments enable experts in a certain domain to actively participate in model-driven development. Developing DSM environments need to be cost-efficient, since they are only used by a limited group of domain experts. Different model-driven technologies promise to allow this cost-efficient development. [1] presented experiences in developing a DSM environment for telecommunication network modelling. There, challenges were identified that need to be addressed by other new modelling technologies. In this paper, we now present the results of addressing one of theses challenges—abstraction and reuse support—with the Reuseware Composition Framework. We show how we identified the abstraction and reuse features required in the telecommunication DSM environment in a case study and extended the existing environment with these features using Reuseware. We discuss the advantages of using this technology and propose a process for further improving the abstraction and reuse capabilities of the DSM environment in the future.

1 Introduction

Domain-specific modelling (DSM) environments enable experts in a certain domain to actively participate in model-driven software development (MDSD)—even if they lack experience in software engineering or software modelling. Since such environments are only used by a limited group of experts, the development needs to be cost-efficient. This can be achieved by developing DSM environments with model-driven technologies instead of implementing them by hand.

[1] presents experiences gained in developing such a DSM environment for telecommunication experts at Telefónica. There, the Graphical Modeling Framework (GMF) [2] was used to develop a graphical editor as core of the environment. [1] identified challenges for technologies that were not met by the tooling used so far. One of the identified challenges is *abstraction and reuse*. That is, supporting domain experts to create abstract views of complex models and to develop reuseable model components.

In this paper, we now report on the results in addressing the *abstraction and reuse* issue with another modelling technology—the Reuseware Composition Framework[3]. Reuseware is founded on Invasive Software Composition [3] and extensions of it [4, 5]. It is based on the Eclipse Modeling Framework (EMF) [6] and integrates into the Eclipse platform. It also has a component for integrating with GMF. Therefore, we used Reuseware to extend the existing domain-specific network modelling environment [1] that is based on Eclipse, EMF and GMF.

The core of the network modelling DSM environment is a domain-specific language (DSL) based on the Common Information Model (CIM) [7] that is a Distributed Management Task Force (DMTF)[4] standard for systems, networks, applications and service definition. The graphical editor of the DSL, which was the main part of the DSM environment as presented in [1], is directly based on an Ecore[5] metamodel that represents a large part of the CIM standard.

In the MODELPLEX project[6], Telefónica defined a case study in which they not only use the CIM-based DSL for telecommunication network modelling, but also formulate abstraction and reuse concerns [9]. Driven by this case study, we developed abstraction and reuse tooling for the DSM environment with Reuseware. During the process, we realised that many design decisions require feedback from the domain experts. Therefore, rapid prototyping and continuous updating of the DSM environment, based on that feedback, is needed. Consequently, we propose a development process for abstraction and reuse features of a DSM environment that is also applicable for other environments than the network modelling tool. We discuss how this development process can be implemented with the model-driven technologies we used.

This paper is structured as follows. Section 2 motivates the need for abstraction and reuse in DSM environments and introduces the network modelling case study. It further shows the features we developed, driven by the case study, with Reuseware. In Sect. 3 we present the development process for improving the DSM environment and Sect. 4. presents conclusions from this work.

2 Abstraction and Reuse Support in the DSM Environment for Network Modelling

In this section, we first motivate the need for reuse and abstraction mechanisms in DSM environments for complex domains. We then show the setup of the case study we conducted and explain the abstraction and reuse features we developed, driven by the case study, for the network modelling tool with Reuseware.

2.1 Reuse and Abstraction in a DSM Environment

A DSM environment is the tooling for a domain-specific language (DSL). A DSL is used to reduce the complexity arising when developing software systems

[3] `http://reuseware.org`
[4] `http://www.dmtf.org`
[5] metamodelling language of EMF; conforms to OMG's EMOF [8] standard
[6] `http://www.modelplex.org`

using a general-purpose language (GPL) such as UML or Java. Unlike a GPL, a DSL focuses on a particular problem domain and contains a relatively small number of constructs that are immediately identifiable to domain experts and allows them to construct concise models capturing the design of the system at an appropriate level of abstraction. While typical DSLs are small languages with a manageable number of concepts, a DSL that embodies a standard vocabulary of a larger domain is a complex language with a large number of concepts. This complexity eventually compromises the very aims against which the DSL was built in the first place: domain focus and conciseness.

The metamodel and the graphical editor [1] developed on the basis of the Common Information Model (CIM) [7] represent such a complex DSL. While this language is a DSL in the sense that it provides dedicated constructs for the telecommunication domain, its size in terms of the number of constructs and features is comparable to that of a GPL such as the UML—the CIM standard defines more than 1500 concepts.

However, the domain for which CIM is designed can be split into more specialised domains where not all details of CIM are required in each of them. The classical MDSD approach would be to construct new DSLs that provide abstractions over and above the constructs provided by CIM. This means that different DSLs, all in the telecommunication domain, are created for different abstraction levels and are combined in an MDSD process.

To employ this approach, one has to identify the abstraction levels and decide which DSLs, and with which constructs, have to be created. This is an iterative process, since a DSL has to be tested and used by the domain experts to evaluate its usefulness and improve it. Updating one DSL alone can be costly when the associated tooling needs to be adapted manually, which is often the case with today's DSL development technology as experienced in the development of tooling for the original CIM-based DSL [1]. This cost would even increase if multiple DSLs, which are connected in an MDSD process, are updated and co-evolved.

Instead of using a classical MDSD approach as described above, we develop abstraction tooling for CIM using model composition. This is done by defining a *composition system* with Reuseware. As we will discuss in Sect. 3, this solution can be used as 1) an alternative for the classical MDSD approach, 2) prototyping for finding the DSLs in the classical approach (and thus avoiding costs of evolution) and 3) basis for implementing the classical approach.

2.2 Case Study Setup

We extended the network modelling DSM environment with abstraction and reuse support driven by a case study defined by Telefónica [9]. For the case study, Telefónica defined a model of a typical ADSL service network configuration for their customers consisting of 52 model elements. An excerpt of the model, displayed in the graphical editor of the DSM environment, is shown in Fig. 1. After the model was defined, the domain experts at Telefónica marked and named parts of the model that can be abstracted into a single concept on a higher abstraction level and reused at different places in the model. For

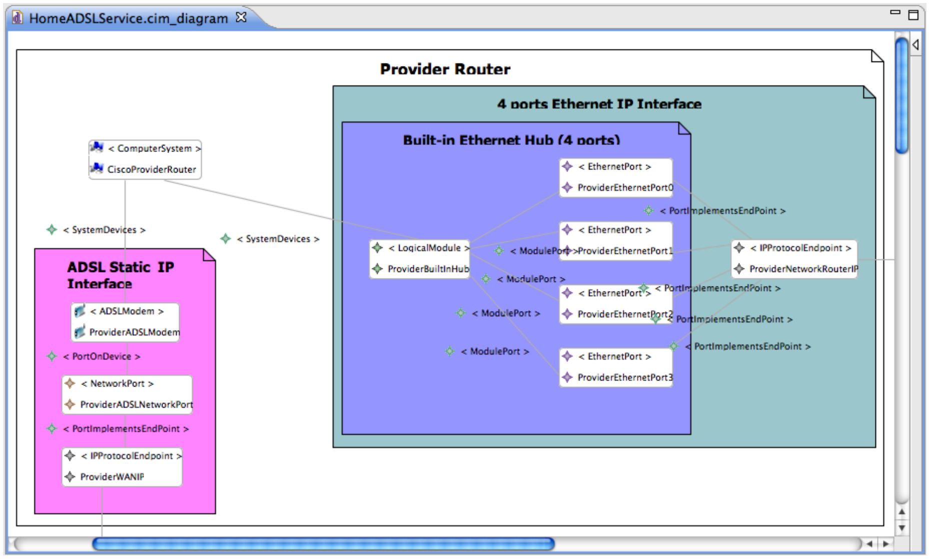


Fig. 1. An excerpt from a CIM model of an ADSL service network topology (provided by Telefónica R&D for the MODELPLEX project [9])

the marking they used notes with different colours. These are only parts of the graphical syntax and do not change the meaning of the underlying model. In addition, for each concept they marked, they provided a list of attributes that need to be visible on a higher abstraction level.

The concepts were then grouped into seven specialised domains (Protocols, Physical Interfaces, Logical Interfaces, Systems, Network Devices, Network Links, Network Topologies) that reside on different abstraction levels (Levels 1–4) as summarised in Fig. 2. For each specialised domain and/or each abstraction level, a separate DSL could potentially be defined. Between the different specialised domains, dependencies can be identified (arrows in Fig. 2). They express which domains from a lower abstraction level are used to express concepts of a higher abstraction level. We note that in principle concepts of Level x can be represented by concepts of Level $x - 1$. However, certain concepts of higher abstraction levels (Level 2, 3 or 4) are also expressed directly with concepts of Level 0. This is, because the CIM standard (Level 0) itself, offers constructs of low (e.g. *EthernetPort*) but also high (e.g. *System*) abstraction.

Note that the levels, domains and dependencies between them as shown in Fig. 2 are the results of a first case study. Using DSM tools based on these results and doing more case studies with more models, will most likely extend and alter the results—which can lead to the mentioned evolution costs when the results are directly manifested in DSLs. Therefore, we present a Reuseware composition system that extends the network modelling DSM environment with abstraction and reuse features as an alternative solution in the following.

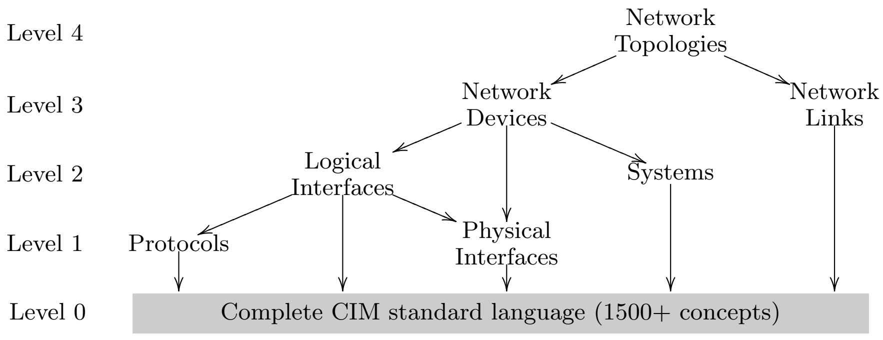


Fig. 2. Abstraction levels and component types for network models

2.3 Analysis and Decomposition of the Case Study Model

We used the Reuseware Composition Framework to define a *composition system* and integrate it into the DSM environment for network modelling. A composition system defines how users of the system—domain experts in our case—can define and compose model components. In Reuseware, a composition system is defined based on an existing DSL to extend it with abstraction and reuse features, while preserving the existing tool support for the DSL. The user is then able to define model *fragments* using the existing editor of the DSL. Such fragments can then be composed graphically by defining *composition programs* in a graphical editor provided by Reuseware with the possibility to reuse one fragment several times in one or different composition programs. Reuseware then interprets the composition programs to merge the fragments to complex models that can again be inspected using the existing DSL editor.

In our case, we defined a composition system for the CIM-based DSL driven by the requirements specified in the case study model (cf. Fig. 1). In the first step, the model was decomposed into fragments following the decomposition suggestions marked in the model. Second, initial composition programs were defined using Reuseware's graphical composition program editor. At this point, the composition programs did not yet contain the composition definitions. Third, we constructed a composition system that allows the composition of CIM fragments in an easy and intuitive way such that domain experts can use it. Finally, we adjusted the fragments to the composition system and completed the composition programs such that they recompose the original case study model.

Figure 4 shows the fragments and composition programs that are the decomposed version of the part of the case study model that is shown in Fig. 1. The three rows in the figure correspond to the abstraction Levels 1–3 (from bottom to top). On Level 1, we have the *BuiltInEthernetHub* fragment, which is a Physical Interface modelled in the CIM-based DSL, and the *IP* fragment, which is a *Protocol* also modelled in the CIM-based DSL. On Level 2, three CIM models are defined. Two Logical Interfaces and one System. The first Logical Interface—*EthernetIPInterface*—is a composition program that contains the

		fragments	avg. no. of model elements	comp. programs	avg. no. of fragments in comp. prgr.	reused
Level 4	NW Topologies	0	n.a.	1	8.00	0
Level 3	NW Devices	0	n.a.	4	3.25	4
	Network Links	2	1.00	0	n.a.	4
Level 2	Logical Interfaces	1	3.00	3	2.67	6
	Systems	1	1.00	0	n.a.	4
Level 1	Physical Interfaces	3	3.00	0	n.a.	4
	Protocols	5	2.60	0	n.a.	7
Total		12	2.42	8	3.63	29

Fig. 3. Fragments of the case study

two Level 1 fragments. On the contrary, the second—*ADSLStaticIPinterface*—is directly modelled in the CIM-based DSL. The *System* fragment is also modelled in the CIM-based DSL. On Level 3, we then have one composition program—*ProviderRouter*—that composes the three Level 2 fragments.

An overview of all fragments and composition programs we obtained by decomposing the complete case study model that consists of 52 elements, is given in Fig. 3. In total, 12 fragments and 8 composition programs were defined. In average, each fragment contains 2.42 model elements which means that a total of 29 model elements were created in the CIM-based DSL. In the original model, 52 elements were modelled, which means that 44% of the case study model can be created by reusing fragments instead of modelling in the CIM-based DSL. To enable domain experts to perform this composition and reuse of CIM model fragments, we defined a composition system with Reuseware that is presented in the following.

2.4 Re-composition of the Case Study Model using the CIM Composition System

Since the composition system for CIM models should extend the existing tooling (the graphical editor of the CIM-based DSL and the Reuseware composition program editor) we introduce five prefixes (`+`, `%`, `?`, `*`, `-`) that can be prepended to attribute values of model elements to define the *composition interface* of a CIM model fragment. A composition interface points at the parts of a model fragment that are exported to be connected with parts of other model fragments in composition programs. The prefixes are explained in the following on the example of Fig. 4.

Looking at the two Level 1 models (Fig. 4; bottom row), we can see that the `BuiltInHub` element in the *BuiltInEthernetHub* fragment is prefixed with `+` and named `+BuiltInHub`. `+` exports the element to the composition interface and lets it appear with the name of the element (in this case `BuiltInHub`). This can be seen in the *EthernetIPInterface* composition program on Level 2 (1st in middle

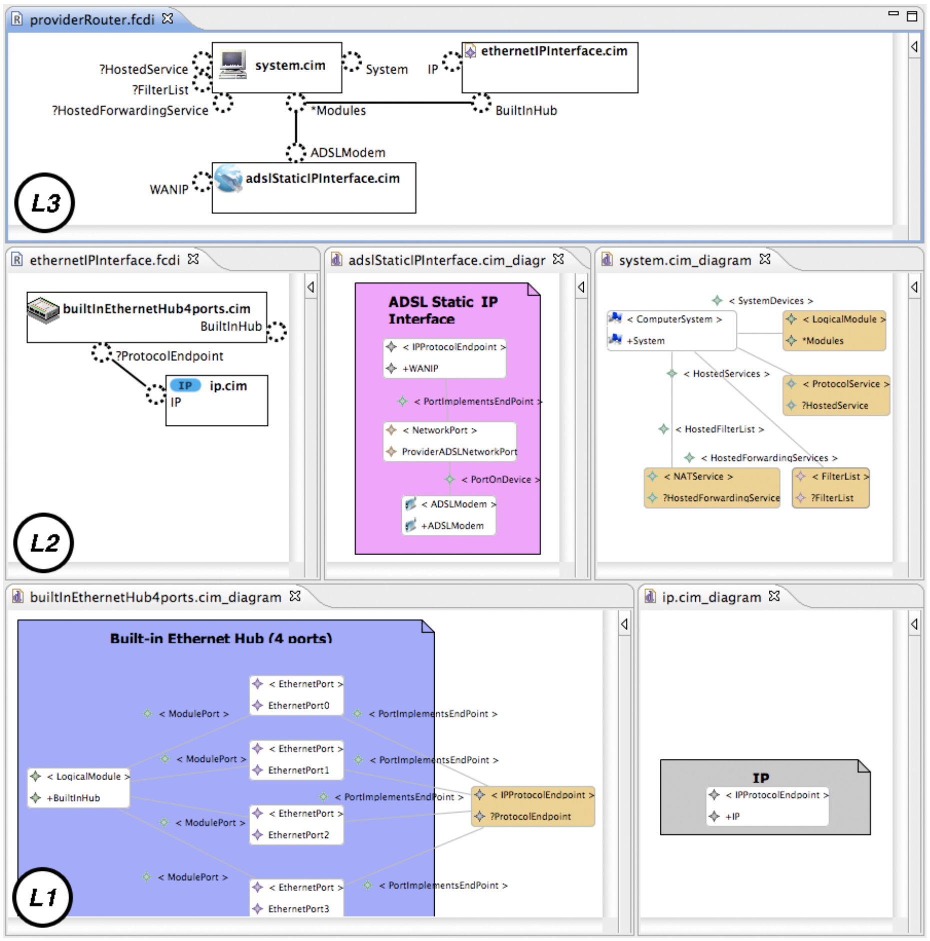
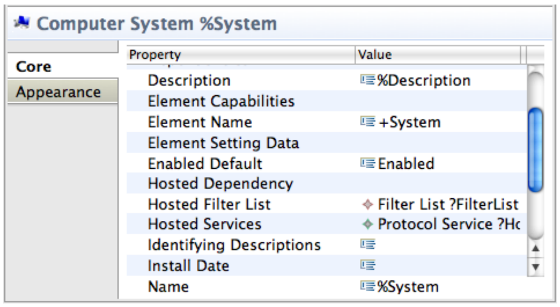


Fig. 4. Fig. 1 decomposed into fragments and composition programs on Levels 1–3.

row). Similar is done with the `IP` element in the *IP* fragment. Furthermore, we add a new element (depicted in orange) to the *BuiltInEthernetHub* that we name `?PrototcolEndpoint` and connect to other elements in the fragment. `?` is used to define a variation point. That is, this is not an element with meaning, but only a placeholder. It will be replaced or removed during composition.

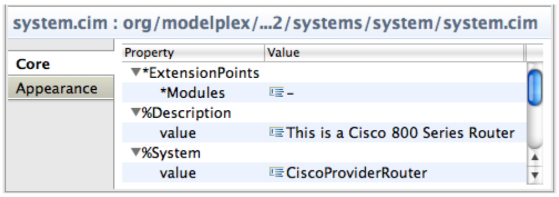
The composition program *EthernetIPInterface* (1st in middle row) on Level 2 can now make use of the composition interface. Concretely, the exported `IP` element is linked to the `?PrototcolEndpoint` variation point. Executing the invasive composition yields an *EthernetIPInterface* fragment that is equal to the corresponding part of the original use case model (cf. Fig. 1). In the *ADSLStaticIPInterface* fragment, we declare two elements to be exported using `+` (`WANIP` and `ADSLModem`). In the *System* fragment, we also export the `System`



Computer System %System

Property	Value
Description	%Description
Element Capabilities	
Element Name	+System
Element Setting Data	
Enabled Default	Enabled
Hosted Dependency	
Hosted Filter List	Filter List ?FilterList
Hosted Services	Protocol Service ?Hc
Identifying Descriptions	
Install Date	
Name	%System

Fig. 5. Properties of the `System` model element (cf. Fig. 4 middle)



system.cim : org/modelplex/...2/systems/system/system.cim

Property	Value
▼*ExtensionPoints	
*Modules	–
▼%Description	
value	This is a Cisco 800 Series Router
▼%System	
value	CiscoProviderRouter

Fig. 6. Properties of the Level 2 fragment *system.cim* (cf. Fig. 4 top)

element, add three variation points (using `?`) and add the extension point **Modules* (using `*`). In contrast to a variation point, an extension point defined by `*` allows for multiple extensions and needs to be explicitly removed by using `-` in a composition program.

Using the exports, variation points and extension points defined in the fragments of Level 1 and 2, the *ProviderRouter* composition program on Level 3 can now be enriched with the composition links that are required to construct a *ProviderRouter* CIM model that is equal to the corresponding part of the original case study model (cf. Fig. 1 top).

Furthermore, the `%` prefix is used in all fragments to export attributes to the composition interface. Figure 5 shows this exemplarily for the `System` element in the *System* fragment (middle row on the right in Fig. 4). We can see that there are many attributes, which is typical for all elements in a CIM model. Therefore, as many attributes as possible should be set to default values if sufficient and only a limited set should be exported to the next abstraction level. Here, we export the Description attribute by setting it to `%Description`. The value followed after `%` defines the name of the attribute on the next abstraction level (here `Description`). The ElementName attribute is also exported to an attribute `System`, because `+` is exported by default. Furthermore, the Name attribute (not to confuse with ElementName) is exported to the attribute `System` (by using `%System`). This means that settings to `System` on the next abstraction level will set both attributes (Name and ElementName) to the same value.

The exported attributes can be modified in the properties of a corresponding fragment instance in a composition program. Figure 6 shows these attributes for the instance of the *System* fragment in the *ProviderRouter* composition program (top row in Fig. 4). We can see that both exported attributes—`%Description` and `%System`—are set. The `%System` attribute (which maps to ElementName on the lower abstraction level) is set to `CiscoProviderRouter` which means that the `+System` element is not re-exported. For re-exporting it, we would need to set the attribute to `+CiscoProviderRouter`. The properties also list all extension points, which is only `*Modules` in this case. An extension point can be removed if it should be no longer visible on the next abstraction level by setting it to `-`, which we do in the case of `*Modules` here.

Another feature to improve the user experience is the specification of icons that are then shown on fragments in composition programs. In the CIM composition system, domain experts can specify icons themselves by placing them next to the fragments they develop.

Using the composition system, we were able to recompose the complete original case study model from the fragments (cf. Fig. 3) that were created from it based on the decomposition proposed by the domain experts (cf. Fig. 1). As mentioned, 44% of the model consists of reused fragments compared to complete manual modelling.

The features of this CIM composition system give the domain experts a lot of freedom. They can introduce new fragments at will and design their composition interface and their look in composition programs individually. They can also introduce new abstraction levels without modifying any language, tooling or composition systems, since all CIM models on abstraction levels higher than 0 are composition programs. Thus, the presented composition system is useful in particular in the early stages of building a DSM environment to find appropriate abstraction levels. Of course, the composition programs expose certain parts of the generic Reuseware tooling. Thus, for certain abstractions it might become desirable to hide more of the composition system and Reuseware to the domain experts, which we will discuss in the following.

3 A Process for Introducing and Improving Reuse and Abstraction in DSM Environments

This paper presented the extension of the DSM environment for network modelling with abstraction and reuse support. For this, we have developed a composition system with Reuseware as a less cost-intensive alternative to developing a set of DSLs for different abstraction levels and connect them via transformations, which would have been the classical MDSD approach. The composition system was developed with relatively little effort—the complete system definition consists of only 103 lines of textual specification in Reuseware specific languages for composition system definition.[7] Nevertheless, the system, as de-

[7] specifications, case study model, fragments and composition programs can be obtained from `http://reuseware.org/index.php/Reuseware_Application_CIM`

scribed in Sect. 2.4, was integrated into the DSM environment and directly used by domain experts. They can now use the additions in the DSM environment to introduce new abstraction levels as required.

Still, there are also (potential) drawbacks in using the developed composition system over the classical MDSD approach. First, the flexibility we gave to the composition system inherently comes with the danger that it again threatens the simplicity and abstraction we wanted to introduce with the it in the first place. Since the domain experts control the composition interfaces themselves to a large degree, they might overload the interface of components or design them too restrictive, which would make fragments hard or impossible to reuse. Second, the tooling (in particular the user interface), which is only a thin layer on top of Reuseware, can never be as highly customised or adjusted to other platforms and technologies as individual DSLs can be.

These drawbacks, however, only apply in certain scenarios. For example, when new users that only work on one particular abstraction level are introduced to the DSM environment often, which justifies the costs of developing customised tools for them; or when people have to work on specific platforms with resource restrictions that can not be met by the Reuseware tooling. To answer such questions for the CIM case, we need to perform more case studies and, most importantly, get feedback from the domain experts on these questions.

Because this feedback from domain experts is of high importance for the whole idea of DSL building, we claim that creating a flexible composition system for an existing complex DSL, as the one shown in this paper for the CIM-based DSL, is a good first step to build abstraction and reuse facilities on top of the existing complex DSL. Even if we switch to a classical MDSD approach later, the composition system is an inexpensive way to obtain a first prototype that can then be used and tested by the domain experts to collect feedback on what the correct abstraction levels are. With this we can obtain the final DSM environment which may consist of the composition system or a set of DSLs (on the correct abstraction levels) or a combination of both.

To support this claim, we propose the following process to continue the development of the DSM environment for network modelling with different abstraction levels that can also be used for developing DSM environments for other domains. The process, consisting of five phases, is visualised in Fig. 7:

1. In the first phase, the developer of the DSM environment collects initial information about desired abstraction levels from the domain experts who are already familiar with the complex existing DSL. In the Telefónica case study this was done in form of markings in the case study model (cf. Fig. 1).
2. In the second phase, the developer designs the first composition system version that is flexible enough to cover all abstraction and reuse requirements identified in step one, but is customised enough such that it can be used by domain experts. For the network modelling DSM environment, this composition system was shown in Sect. 2.4.
3. In the next phase, the domain expert uses the composition system, creates fragments and gives feedback. The developer can give support in this phase.

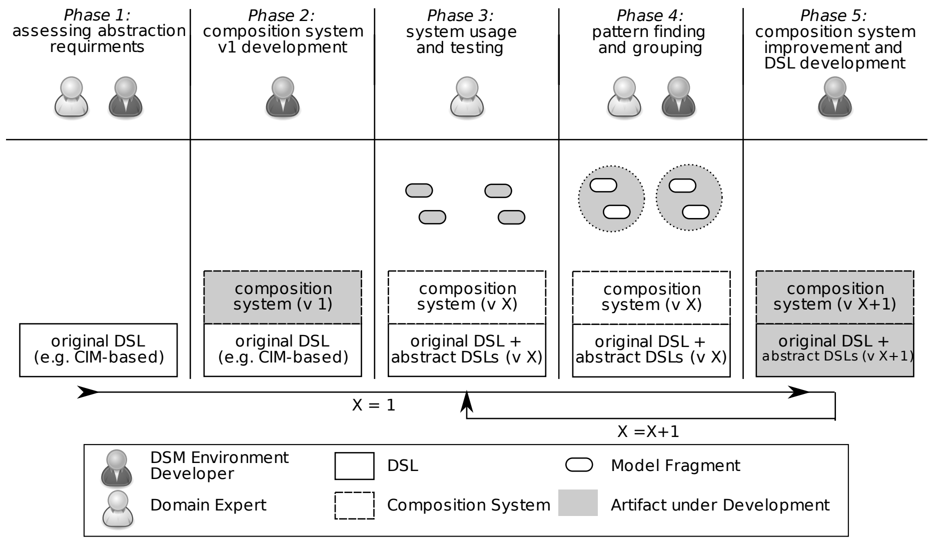

Phase 1: assessing abstraction requirments
Phase 2: composition system v1 development
Phase 3: system usage and testing
Phase 4: pattern finding and grouping
Phase 5: composition system improvement and DSL development
original DSL (e.g. CIM-based)
composition system (v 1)
original DSL (e.g. CIM-based)
composition system (v X)
original DSL + abstract DSLs (v X)
composition system (v X)
original DSL + abstract DSLs (v X)
composition system (v X+1)
original DSL + abstract DSLs (v X+1)
X = 1
X =X+1
DSM Environment Developer
Domain Expert
DSL
Composition System
Model Fragment
Artifact under Development

Fig. 7. Process for obtaining a DSM environment for a complex DSL

We started this phase for the CIM-based DSL when we created the fragments and composition programs in Sect. 2.3.

4. In phase four, the fragments and composition programs are analysed to find common patterns and to group the fragments following these patterns. From these patterns, the developer can derive restrictions and default behaviour for the composition system or identify constructs for abstract DSLs and refine the abstraction levels. In the CIM composition system for instance, it turned out that `EndPoint` elements (used e.g. several times in Fig. 4) are nearly always exported because they mark places where components are connected in the physical world. Thus one useful refinement of the composition system might be to export `EndPoint` elements by default.
5. In the last phase, the developer improves the composition system based on the results from the previous phase or builds abstract DSLs that replace (parts of) the composition system. The result is then given to the domain experts. It is either the final DSM environment or a next prototype and phase three to five are repeated.

If the abstraction levels are manifested and the customisation capabilities of the composition system are not sufficient, the developer can decide to build abstract DSLs with other technologies. One way he can go to keep the costs of this low is to use generative technologies for the DSL tooling such as EMFText [10] or EuGENia [11] that allow the generation of textual or graphical editors with minimal effort. Transformations between the new DSLs can be realised using a model transformation approach or reusing the already existing composition system. We presented this idea in [12].

4 Conclusion

In this paper, we used Reuseware to add abstraction and reuse support to a domain-specific modelling environment for telecommunication networks. With this, we addressed one of the missing features of the environment identified in [1]. Judging by the case study we performed, Reuseware provides the necessary means to extend the environment with the desired features with acceptable effort.

Performing the case study, we realised that several iterations for the DSM environment are necessary to find which abstraction and reuse features make the domain experts' work most efficient. Therefore, we proposed an iterative development process that focuses on collecting feedback from domain experts and performing rapid prototyping with Reuseware. In the future, we need to realise this process in larger case studies to improve the DSM environment.

5 Acknowledgments

This research has been co-funded by the European Commission in the 6th Framework Programme project Modelplex contract no. 034081 (www.modelplex.org).

References

1. Evans, A., Fernández, M.A., Mohagheghi, P.: Experiences of Developing a Network Modeling Tool Using the Eclipse Environment. In: Proc. of ECMDA-FA'09. Volume 5562 of LNCS., Springer (June 2009)
2. Gronback, R.C.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Pearson Education (April 2009)
3. Aßmann, U.: Invasive Software Composition. Springer (April 2003)
4. Heidenreich, F., Henriksson, J., Johannes, J., Zschaler, S.: On Language-Independent Model Modularisation. In: Transactions on Aspect-Oriented Development VI. Volume 5560 of LNCS., Springer (October 2009)
5. Henriksson, J.: A Lightweight Framework for Universal Fragment Composition—with an application in the Semantic Web. PhD thesis, TU Dresden (January 2009)
6. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: Eclipse Modeling Framework, 2nd Edition. Pearson Education (January 2009)
7. Distributed Management Task Force Inc. (DMTF): Common Information Model Standards. `http://www.dmtf.org/standards/cim` (January 2010)
8. Object Management Group: MOF 2.0 Core Specification. `http://www.omg.org/spec/MOF/2.0` (January 2006)
9. MODELPLEX Project: Deliverable D1.1.a (v3): Case Study Scenario Definitions. `http://www.modelplex.org` (March 2008)
10. Heidenreich, F., Johannes, J., Karol, S., Seifert, M., Wende, C.: Derivation and Refinement of Textual Syntax for Models. In: Proc. of ECMDA-FA'09. Volume 5562 of LNCS., Springer (June 2009)
11. Kolovos, D.S., Rose, L.M., Paige, R.F., Polack., F.A.: Raising the Level of Abstraction in the Development of GMF-based Graphical Model Editors. In: Proc. of 3rd MISE Workshop @ ICSE. (May 2009)
12. Johannes, J., Zschaler, S., Fernández, M.A., Castillo, A., Kolovos, D.S., Paige, R.F.: Abstracting Complex Languages through Transformation and Composition. In: Proc. of MoDELS'09. Volume 5795 of LNCS., Springer (October 2009)