# The power of MOF-based meta-modeling of components [*]

Petr Hnetynka[1], Frantisek Plasil[1,2]

[1]Department of Software Engineering
Faculty of Mathematics and Physics
Charles University, Malostranske namesti 25
Prague 1, 118 00, Czech Republic

[2]Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod Vodarenskou vezi 2, Prague 8
182 07, Czech Republic

{*hnetynka, plasil*}@*dsrg.mff.cuni.cz*

## Abstract

*To allow comfortable and easy development, component systems have to provide a rather a big set of development supporting tools including at least a tool for composition and repository for storing and retrieving components. In this paper, we evaluate and present advantages of using MOF and meta-modeling during definition of component system and also during development of the supporting tools. Most of the presented arguments are based on a broad practical experience with designing the component systems SOFA and SOFA 2; the former designed in the classical ad hoc "manual" way, while the latter via meta-modeling.*

## 1 Introduction

Component-based development (CBD) has become a well-understood and widely used technique for developing not only large enterprise applications, but in fact for any type of applications, including embedded ones. Using this technique, applications are built by composing already developed components. Every component system (i.e. a system and/or framework allowing to develop and compose components) uses a different view as to what a software component is, but a generally agreed consensus is that "component" means a black-box entity with well-defined interface and behavior. The interface of a component comprises the services provided by it and the services required from other cooperating components and/or an environment (container). To specify its particular view on components, a component system defines its component model, i.e. a set of abstractions, which together define components, their composition, etc. Thus, the term component has to be always interpreted in the scope of a given component model.

In order to allow really fast and comfortable development and management of component-based applications, component systems should provide rather a big set of development supporting tools and infrastructure. These tools and infrastructure usually comprise of at least a tool for developing and composing components and a repository storing and serving already developed components.

However, creating such an infrastructure is rather tedious and time-and-other-resources-consuming task. This is probably why especially academia-based component systems provided sophisticated component models with plenty of advanced features, but with no or very limited support for real development of components at a large scale. In order to overcome this problem, modern component systems try to heavily employ modeling and meta-modeling approaches that allow automatic generation of many supporting tools.

The component models of classical ("old") component systems were usually defined by an ADL (Architecture Definition Language). Since these ADL languages were proprietary, the development tools were developed completely manually from scratch. Another related problem was that the semantics of a component model had to be typically defined in a natural language. Finally, as the cores of component models had been very similar (in many case in fact the same), a straightforward idea was to allow interoperability between models and use component from one model in another. This issue is not only making components exchangeable between the systems but also it requires interoperable tools to allow developing and managing such heterogeneous applications. But with hand-made tools and infrastructure, the interoperability was quite difficult.

As stated above, the modern component systems usually use meta-modeling approaches to define their component models and, more interestingly, to automatically generate repositories, tools for development, editors for designing and composing components, etc.

Additionally, meta-models provide means for defining semantics in a formal way (at least partially), and also there

[*]This work was partially supported by the Czech Academy of Sciences project 1ET400300504.

are approaches supporting easy interoperability and transformations between different models.

All these advantages of the meta-modeling approaches bring faster development and maintenance of the component systems themselves and therefore faster adoption of the systems to the production.

### 1.1 Goal and structure of the paper

In this paper, based on our experience with designing and developing component systems (SOFA in particular) and analysis of several existing component systems, such as Fractal and Koala, we present the advantages of meta-modeling approach in component systems. Also we evaluate and compare the meta-modeling approach with the classical one by comparing the SOFA [22] (based on ADL) and SOFA 2 [7] (based on meta-model) component models. To achieve the goal, the paper is structured as follows. Section 2 presents an overview of meta-modeling principles and contemporary component models. In Section 3, we articulate advantages of using meta-models for component systems specification, design, and implementation, while Section 4 compares SOFA and SOFA 2 definitions and also presents related work. Section 5 concludes the paper.

## 2 Background

### 2.1 Models and meta-models

Models and meta-model are the main concept in MDD (model-driven development), which is one of the most popular development paradigms nowadays. In MDD, a system is developed as set of models. A typical approach starts with modeling the system on a platform independent level, i.e. capturing only the business logic of the system, leaving out any implementation details. In a series of transformation, this platform independent model is then altered into a platform specific model, i.e. a model reflecting also the details specific for the platform chosen for implementation (such as Java and .NET).

The abstractions featuring in a model M, i.e. the elements to be used for modeling, are described by a meta-model, i.e. a model of M. Many times, the meta-model is referred to as a domain specific language, since it defines a means for modeling system in a specific domain.

An important standard employing the idea of MDD is the OMG's Model Driven Architecture (MDA) [16] specification. For describing meta-models, OMG has defined the Meta-Object Facilities (MOF) [17] standard. It defines a language and framework for specifying and constructing meta-models and for implementing repositories holding instances of models. To enable interchange of these instances among repositories, MOF defines an XML based format denoted XMI (an abbreviation of "XML Meta-data Interchange").

Since the MOF language does not have any specific visual representation, for specifying meta-models, a subset of UML class diagrams is used. As an aside, since the versions 2.x of both UML and MOF were introduced, there is a common core of the MOF and UML meta-models.

In the rest of the paper, by "meta-model" we always mean a MOF-based meta-model.

The primary meta-model elements are the class (defining abstractions), and association[*] (defining relations among classes). For illustration, Figure 1 shows a tiny subset of the core of the SOFA 2 meta-model (details are explained in Section 3.1).

### 2.2 Component models

As mentioned in Section 1, to allow really fast and efficient application development via component composition, a component system has to provide a rather large set of development and management tools. These tools should provide at least a functionality for component composition (defining the architecture of an application composed of components) and a repository to store the already available components (both those designed ad hoc to fulfill a very specific task, and generic ones intended for reuse).

Classical component systems like Darwin [14], Wright [1], and others are defined around their ADL. In fact, an ADL determined the corresponding component model via the syntactical constructs corresponding to particular abstractions and their relation. The semantics was described in plain English. These systems did not provide any repository and thus they were intended for capturing the architecture and component composition, not providing means for component reuse. ACME [9] was an attempt to create a common ADL (de-facto standard) but it was not widely adopted. Tools for all these ADLs were developed manually ad hoc and usually allowed to reason about correctness of behavior composition.

Contemporary component systems usually provide more complex infrastructure since they are not ADL centered. In the following overview we focus on several component systems which deal with the whole component lifecycle (from design to run-time).

An industry-based system is the CORBA Component Model (CCM) [15]; it is based on a flat component model, i.e. components cannot be composed hierarchically. Components are described in CORBA IDL v.3 and composition is done at run-time (i.e. there are no IDL constructs for that).

[*]To be precise, they should be named meta-classes and meta-associations (as they express meta-model) but for the sake of simplicity, we use only terms classes and associations in the paper.

The run-time infrastructure is defined by the OMG deployment and configuration specification and offers a repository, deployment environment, etc. — all of them described by meta-models.

The Koala component model [19] is the based of another industrial system and allows creation of hierarchical component-based embedded applications. It uses its own ADL (heavily inspired by Darwin) and the developed components are stored in a repository for further reuse. Also, there are tools for visualization of architectures. The whole infrastructure, including an ADL compiler and repository were designed ad hoc and written manually.

Fractal [4] is an abstract component model, which has multiple implementations (in Java, C, . . . ). Components are primarily built at run-time using API calls (defined in the Fractal specification). An ADL also exists, but it can be viewed as a "shortcut" for creation components and architectures — all definitions of components are transformed into the API calls. The specification of Fractal does not prescribe existence of any repository but there are attempts to do so.

The SOFA component model [22] and, especially, its new version SOFA 2 [7] furnishes a general purpose component system. The original version of SOFA was defined by ADL and the repository and all tools were written by hand while SOFA 2, has been redesigned using a meta-model and most of its supporting tools and the repository have been generated from the meta-model. In a more detail, the comparison between these versions is discussed in Section 4.

The list of component system above definitely is not complete (especially the list of those designed in academia is not a shore one) but it provides a base for comparing SOFA with those designed at least partially for industrial applications. There are also other contemporary component systems defined by a meta-model, for example Palladio [3], PRISMA [21], and others, but most of them do not support the whole component lifecycle (usually they focus on design only). For more details see Section 4.

## 3 Applying MOF in component model design and implementation

As mentioned in Section 1 and Section 2.2, many contemporary component systems are converting to or have been defined from the beginning by a meta-model. From these component systems, most of them use for their meta-model definition and repository generation the Eclipse Modeling Framework (EMF) [8], which is an implementation of the MOF standard (even though EMF does not rigorously comply with the MOF standard the differences are very subtle, not visible to the developer using EMF). In addition to being a mature tool well supported an maintained, EMF is popular since it is available freely as an open source plugin for Eclipse, and also since a framework for generating model editors from a meta-model definition (called GMF — Graphical Modeling Framework [10]) is available.

In summary, based on experience with SOFA 2, the three most important benefits of using meta-models are (i) the definition of the component model syntax but also its semantics, (ii) the relatively fast and semi-automated creation of the development supporting tools, and (iii) the semi-automated creation of runtime management tools. In the rest of this section, we go through these topics in more details.

### 3.1 Defining semantics of a component model

In a component model specification, the meta-model defines the elements and abstractions (and also the associations among them) forming the component model. Importantly, by the associations, it explicitly defines the relations among these abstractions.

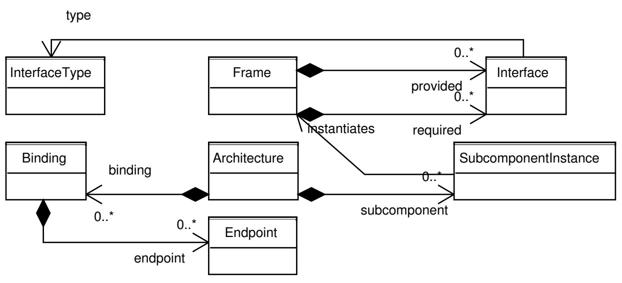

**Figure 1. A core of the SOFA 2 meta-model**

Figure 1 shows an example of a meta-model, which is a core subset of the meta-model of the SOFA 2 component system. It defines there basic abstractions of SOFA 2: component type (called Frame), component implementation (called Architecture), and interface type. A frame provides black-box view of a component instance by defining its provided and required interfaces, which are defined using interface type. The frame is implemented by an architecture, which can contain subcomponents instances (defined again by frames) and bindings among these subcomponents. If the architecture does not have subcomponents, any component instance defined by the frame is a primitive component the implementation of which is to be provided in an underlying programming language.

Via associations, the meta-model in an easy-to-comprehend way determines (at least partially) the semantics of the component model; obviously, there is a part of the semantics that has to be defined informally in plain English, e.g. "Frame is a component type".

In addition to the classes and associations, relations can be more precisely defined by constraints added to the classes and associations. These constraints are typically expressed by OCL (Object Constraint Language). A particular constraint in the SOFA 2 core might be, for example: `self.subcomponent->forAll(s1,s2 | s1.name<>s2.name)`, meaning that the names of subcomponents in the architecture determined by self have to be distinct.

To summarize, meta-model and constraints provide a complex, but simple to use and understand, standard means for expressing both the abstractions of a component model and also their semantics in a formal way. On the contrary, a component system defined by ADL requires its semantics to be defined mostly in plain English and is thus more likely ambiguity and error prone.

### 3.2 Infrastructure creation

Second, and even more important advantage of meta-modeling approach is the simple creation of the infrastructure (tools) for component development. As we already noted, the tools are crucial for a successful adoption of a component system (below we assume that is base on a component model CM).

Once the meta-model CMM of CM has been defined, it is very easy to create the following development infrastructure tools. First, the repository storing designed components (i.e. instance of CM) can be generated completely automatically. This is one of the main functionalities provided by MOF. Another "for-free" obtained functionality is the option to interchange data among such repositories in the XMI format.

Furthermore EMF can automatically generate a semi-visual editor for CM. The editor is very simple but it enforces maintaining the relations among abstraction by allowing to create and connect only correct elements in the way compliant with the meta-model of CM.

Moreover, an EMF extension GMF can generate a more sophisticated visual editor for CM. In addition to CMM, the developer specifies the style of visual representation of each abstraction (element), and then the most of the editor is automatically generated as an Eclipse plugin (the plugin can be launched as a part of the Eclipse IDE, or configured to run as standalone application built over the core of Eclipse). There are still functionalities that have to be designed and developed by hand (like connections to the repositories), but the visual part of the editor is fully generated.

Another contribution to development infrastructure creation is the potentially easy interoperability between component systems. As the component definitions are stored in the generated repositories and a repository can export these definitions in XMI, a transformation between component models CM1 and CM2 can be done very easily, assuming they are based on similar abstractions. This way, e.g., a component designed for CM1 can be transformed and reused in CM2. Moreover, OMG already provides a standardized language called QVT (Query-View-Transform) [18] for performing queries in and transformations on models. QVT by itself does not automatically transform component definitions, but it provides means for an easy and standardized definition of a transformation.

### 3.3 Creation of runtime management tools

In a similar way as the tools for component development, tools for runtime management can be generated. This is particularly important for a component system which allows creating distributed application, where it is necessary to deploy component instances into particular run-time nodes.

Again, to define deployment environment model EM (based on abstractions for e.g. set of hardware nodes, their interconnections, their properties like current load and memory usage), a meta-model EMM has to be defined. Then a repository can be generated from it and for instance used by (i) an environment monitoring tool which feeds the repository with the current status of the environment (current instance of EM), by (ii) a deployment tool which based on the requirements of the deployed application and the current environment status (current instance of EM) creates deployment plan, and by (iii) a GMF visualization tool helping to observe the current status of the environment. Moreover, EMM can include a meta-model of deployment descriptor to allow an automated generation of an EMF/GMF editor.

And again as in the case of development tools, having EMM yields the benefit of achieving relatively easily deployment interoperability among the component systems based on similar abstractions.

## 4 Evaluation and related work

*Evaluation.* In essence, we have so far argued that with the MOF-based meta-models the designers and developers of a component system can focus mainly on the definition of a component model, its abstractions, relations, etc., and the "boring" parts of the implementation can be automatically generated. To justify our claims about the advantages of MOF-based meta-models, we present a brief comparison of two version of the SOFA component system: the original one (further "old SOFA") defined by ADL, and the new one (SOFA 2) defined by a meta-model. Basically, this section is a substantial extension of the comparison we published in [13], where we focus only on the process of implementing repositories.

As described in Section 2.2, old SOFA was specified by a definition of its ADL, which had a CORBA IDL — like structure, with constructs added for describing component types and implementations (architectures). The other necessary descriptors (like deployment descriptor) had also a proprietary structure. The repository for storing all the old SOFA model elements was developed completely by hand. As mentioned in [13], the development of the repository took approximately four person-months. A significant amount of additional time was spent on debugging the implementation. The semantics of the component model was defined only by description in plain English. As SOFA evolved and new features were added, each of such additions and/or changes (e.g. the introduction of software connectors for communication among components) required hand-made changes in the implementation of the repository and also of all related tools, and again took rather nontrivial amount of time for debugging.

On the contrary in the case of SOFA 2 — based on experience gained during development and usage of old SOFA — the ADL-based definition of the component model has been replaced by an EMF-based meta-model. Currently, all the SOFA 2 semantics which could not be expressed via EMF is specified in plain English. Applying OCL constraints is left as a near-future work.

With the aim to emphasize the positive experience with semi-automated generations of the SOFA 2 development and deployment supporting tools from the EMF-based meta-model, we provide below a brief overview of the gained benefits:

(1) Component repository — its development took only one person-month and most of the time was spent on designing and tuning the meta-model and the actual repository was generated within few seconds (only the layer providing remote access to the repository was written by hand).

(2) For developing SOFA 2 components, a GMF-based visual tool proved to be essential. Naturally, the core part of the tool is generated from the meta-model. The components developed by the tool are directly stored into the repository. As an aside, developing components via ADL is still possible, but this is intended as a supplementary option (it was employed during the initial stages of the SOFA 2 development when the visual tool was not ready). Nevertheless, since ADL can be interpreted as another SOFA2 meta-model, particular frame and architecture specification in ADL (model instances) are transformed via XSLT and then directly fed into the repository. An intention is to apply QVT (Sect. 3.2) for this purpose in the future.

(3) As to runtime management tools, an EMF-based meta-model of the deployment plan was designed and via GMF a corresponding visual editor was generated. Even though additional tools featuring the functionality mentioned in Section 3.3 will be subject to future work, the flexibility gained by the existence of the meta-model in terms of generating these tools is incomparable with the ad hoc formed, hard-to-maintain deployment supporting tools of old SOFA.

In addition to automated generation of the supporting tools mentioned above, we identified the following improvements of the meta-modeling approach in SOFA 2 over the "classical way" the old SOFA was designed.

(i) The key advantage we very much appreciate has been the lucidity of the meta-model allowing to immediately see the context and consequences of a proposed modification; this very much helps with achieving and maintaining the component model consistency.

(ii) Most of the changes to the SOFA 2 component model mean only updating the meta-model and then a regeneration of the repository (and other tools).

(iii) The definition of the meta-model significantly reduces the time required to understand the SOFA 2 component model; usually it is sufficient only to show the meta-model to a person familiar with commonly CBD used concepts and, because SOFA 2 uses those as well, he/she immediately understands details of the SOFA 2 component model. This proved to be quite important and beneficial during our participation in a joint project [23] while sharing details on SOFA 2 with our partners

(iv) As to transformation between component models, we have done a simple Fractal ADL to SOFA2 ADL transformation based on the XSLT format. This way, we achieved relatively easily the reuse of several Fractal components, developed for the CoCoME contest application [5], in the SOFA 2 version of the contest application [6]. Once a Fractal meta-model is available (see below), a QVT-based transformation could be created for this purpose.

As the only potential disadvantage we see the fact that the repository interface is generic and, therefore, less intuitive than a single purpose, hand-written one [13]. On the other hand, the generic interface following standards can bee seen as an advantage, since the generic clients for the repository available elsewhere can be reused.

Overall, compared to old SOFA, the definition of the SOFA 2 meta-model in EMF was a big step forward, in terms of the component model design and specification, implementation of supporting development tools, and deployment environment design.

*Related work.* To our knowledge, few contemporary component systems have been defined by a MOF-based meta-model. One of them is the Palladio component model [3]. The model is completely defined in an EMF-based meta-model. Also all the tools for defining and composing components at the architecture level and also the simulation tools are generated with the help of GMF. Another one is the PRISMA component system [21], which core is also described using EMF and development tools are again generated. The component model of Fractal [4] is primarily defined by Java API with semantics description in plain English (Section 2.2). Recently, initiatives to design a meta-model of Fractal were announced with the aim to exploit potential benefits (an initial version of the meta-model and tools can be already found in the Fractal SVN [12]).

Also outside the software components community, strong tendencies to move to the meta-model-based definition can be currently witnessed. First, there is a bunch of MDA-based frameworks for developing applications. An example of such a framework is AndroMDA [1]. Another approach of employing meta-models is used in Software Factories (SF) [11]. For specifying meta-models and DSLs, SF do not use specifically MOF, but the overall approach is similar. A similar functionality is provided by openArchitectureWare [20]. It is a modular MDA/MDD framework built with help of EMF.

## 5 Conclusion

In this paper, based on our experience gained while designing and developing the component systems SOFA and SOFA 2 and also participating in the Q-ImPrESS international project, we discussed and presented the power of MOF-based models and meta-models applied in designing component systems. We argued that its usage significantly reduces the time necessary to develop supporting tools. Advantageously, since the interfaces of these tools follow standards, it is much more easy to provide interoperability among different component systems and their tools. The key advantage we experienced and appreciated was the lucidity of the meta-model, allowing to immediately see the context and consequences of any proposed modification; this very much helps with achieving and maintaining the component model consistency. This is mostly because the meta-model formally in an easy-to-read and comprehend way defines the semantic relations among the component model abstractions — in many cases no additional description in plain English is required.

## References

[1] Allen, R.: A Formal Approach to Software Architecture, PhD thesis, School of Computer Science, Carnegie Mellon University, 1997

[2] AndroMDA, `http://galaxy.andromda.org/`

[3] Becker, S., Koziolek, H., Reussner, R.: Model-Based Performance Prediction with the Palladio Component Model, Proc. of WASP 2007, Buenos Aires, Argentina, Feb 2007

[4] Brunneton, E., Coupaye, T., Stefani, J.B.: Recursive and Dynamic Software Composition with Sharing, Proc. of WCOP'02, Malaga, Spain, Jun 2002

[5] Bulej, L., Bures, T., Coupaye, T., Decky, M., Jezek, P., Parizek, P., Plasil, F., Poch, T., Rivierre, N., Sery, O., Tuma, P.: CoCoME in Fractal, Chapter in The Common Component Modeling Example: Comparing Software Component Models, LNCS, Apr 2008

[6] Bures, T., Decky, M., Hnetynka, P., Kofron, J., Parizek, P., Plasil, F., Poch, T., Sery, O., Tuma, P.: CoCoME in SOFA, Chapter in The Common Component Modeling Example: Comparing Software Component Models, LNCS, 2008

[7] Bures, T., Hnetynka, P., Plasil, F.: SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model, Proc. of SERA 2006, Seattle, USA, IEEE CS, Aug 2006

[8] Eclipse Modeling Framework, `http://eclipse.org/emf`

[9] Garlan, D., Monroe, R. T., Wile, D.: Acme: Architectural Description of Component-based systems, In Foundation of Component-based Systems, Cambridge Univ. Press, 2000

[10] Graphical Modeling Framework, `http://eclipse.org/gmf`

[11] Greenfield, J., Short, K., Cook, S., Kent, S.: Software factories: assembling applications with patterns, models, frameworks and tools, Wiley publishing, 2004

[12] Fractal website, `http://fractal.ow2.org/`

[13] Hnetynka, P., Pise, M.: Hand-written vs. MOF-based Metadata Repositories: The SOFA Experience, Proc. of ECBS'04, Brno, Czech Rep., IEEE CS, May 2004

[14] Magee, J., Kramer, J.: Dynamic Structure in Software Architectures, Proc. of FSE'4, San Francisco, USA, Oct 1996

[15] OMG: CORBA Components, v 3.0, formal/02-06-65, Jun 2002

[16] OMG: Model Driven Architecture (MDA), ormsc/01-07-01, Jul 2001

[17] OMG: MOF 2.0 Core, ptc/03-10-04, Oct 2004

[18] OMG, MOF QVT, ptc/07-07-07, Jul 2007

[19] van Ommering, R., van der Linden, F., Kramer, J., Magee, J.: The Koala Component Model for Consumer Electronics Software, IEEE Computer, Vol. 33, No. 3, pp. 78-85, Mar 2000

[20] OpenArchitectureWare, `http://www.openarchitectureware.org/`

[21] Perez, J., Ali, N., Carsi, J. A., Ramos, I.: Designing Software Architectures with an Aspect-Oriented Architecture Description Language, Proc. of CBSE'06, Vasteras, Sweden

[22] Plasil, F., Balek, D., Janecek, R.: SOFA/DCUP: Architecture for Component Trading and Dynamic Updating, Proc. of ICCDS'98, Annapolis, USA, May 1998

[23] Q-ImPrESS, `http://www.q-impress.eu/`