Applying Megamodelling to Model Driven Performance Engineering

Mathias Fritzsche
SAP Research CEC Belfast
and Queens University Belfast
United Kingdom

Hugo Bruneliere
AtlanMod Team
Ecole des Mines de Nantes
France

Bert Vanhooff
and Yolande Berbers
DistriNet, K.U.Leuven
Belgium

Frédéric Jouault
AtlanMod Team
Ecole des Mines de Nantes, France

Wasif Gilani
SAP Research CEC Belfast
United Kingdom

Abstract

Model Driven Engineering (MDE) has to deal with an increasing number of interrelated modelling artefacts. The Model Driven Performance Engineering (MDPE) process is one concrete illustration of such a situation. This process applies MDE within the context of performance engineering in order to support domain experts, who generally lack the necessary performance expertise. In this paper, we demonstrate the use of megamodelling to manage the numerous artefacts involved in MDPE. Megamodelling enables the explicit modelling of the metadata on MDE artefacts, including possible relationships between those artefacts. Appropriate tool support enables different stakeholders to exploit this additional information. Applying the megamodelling to MDPE pointed out the need for an extension of the existing approach. Thus, the result of the paper is twofold: first, an extension of megamodelling is proposed, second the benefits of the approach are shown on the MDPE use case. We claim that the extension is not solely useful for the latter case, but has a more generic applicability.

1 Introduction

Model Driven Engineering (MDE) focuses on models as first-class artefacts in the software development process. Many different types of models in the broad sense (e.g., UML, ER, etc.) can be handled and several kinds of operations can be applied to them (transformation, code generation, weaving, etc). The exact combinations of, and underlying relations among all these MDE artefacts may depend on the company involved, the project, the date and time, etc.

Several sub-areas of MDE such as model storage (e.g., the EMF [1] registry), transformation specification and execution (e.g., ATL [2], QVT [3]), model weaving (e.g., AMW [4]) and code generation have already been thoroughly explored. For many of them, there are mature and well-established tools available. A common and important characteristic of all these tools is that they focus only on one well-contained MDE area at a time. In a realistic MDE environment, however, we need to combine many types of modelling artefacts and many different MDE techniques (concerning different MDE areas) in order to establish a complete MDE process. For example, we might need to specify a transformation chain that consumes different types of models, simultaneously produces traceability information and finally generates source code. Thus, an important question that arises is how we can handle such a potentially large number of MDE artefacts and manage their associated metadata in order to be able to integrate many different tools and techniques in a seamless way.

In this paper, Model Driven Performance Engineering (MDPE) is first presented in Section 2 as a real-life MDE process where many different types of models and MDE techniques are combined. Within this context, we identify concrete difficulties that are encountered when trying to deal with the numerous involved MDE artefacts. In Section 3 the concrete need for *Megamodelling* (or *modelling in the large*) is identified, its basic principles are introduced and proposed as a suitable solution to solve some of the current difficulties of the MDPE approach.

In Section 4, we shown how we can define an extension of the AM3 megamodelling tool to deal with real-life MDE processes such as the MDPE process. The proposed Megamodelling-based solution and the first results obtained are provided in Section 5. Related work is described in Section 6 and we wrap up with a conclusion and future research directions of the presented work (Section 7).

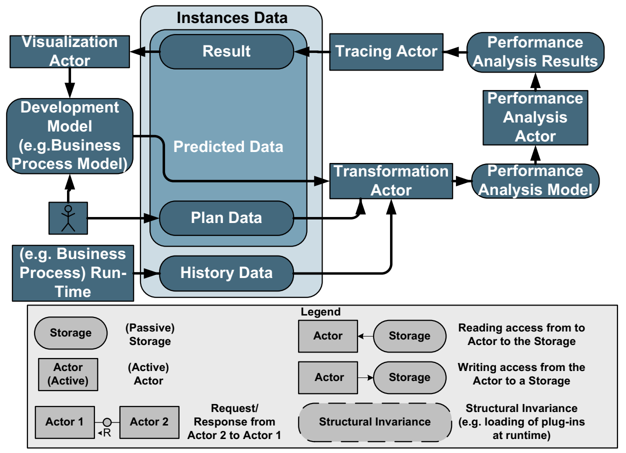

Instances Data
Visualization Actor
Result
Tracing Actor
Performance Analysis Results
Development Model (e.g.Business Process Model)
Predicted Data
Performance Analysis Actor
Transformation Actor
Performance Analysis Model
Plan Data
(e.g. Business Process) Run-Time
History Data
Legend
Storage
(Passive) Storage
Actor (Active)
(Active) Actor
Actor 1
R
Actor 2
Request/ Response from Actor 2 to Actor 1
Actor
Storage
Reading access from to Actor to the Storage
Writing access from the Actor to a Storage
Structural Invariance
Structural Invariance (e.g. loading of plug-ins at runtime)

Figure 1. Model Driven Performance Engineering as a block diagram [5]

2 Model Driven Performance Engineering

We have defined Model-Driven Performance Engineering (MDPE) in previous work [6] as an application of MDE to the performance engineering field. The process offers performance-related[1] decision support to domain experts [8] based on the process models (e.g. BPMN[9]) they use. We refer to this category of models as *Development Models*.

Figure 1 shows basic ideas behind MDPE. Development Models are transformed to *Performance Analysis Models*. For this transformation, additional data about previously processed instances of the input model, such as historical resource demand (see *History Data* in Figure 1) and data about future instances (e.g. a future workload, see *Plan Data* in Figure 1), have to be taken into account. The *Performance Analysis Results* can be traced back and visualized based on the original models. These results are performance predictions for future instances of the modelled process. In the implementation of MDPE, several different modelling artefacts are involved to capture all required information. In this section we give an overview of MDPE and describe some of its inherent problems.

2.1 MDPE Models

In the MDPE process we distinguish between *Development Models* and *Performance Analysis Models* (see Figure 2). Development Models, such as BPMN models [9], are used as development artefacts by the domain expert, i.e. the business domain expert. In the MDPE process, the development models are transformed to Performance Analysis

[1]Performance is defined as "the degree to which an application or component meets its objectives for timeliness" [7].

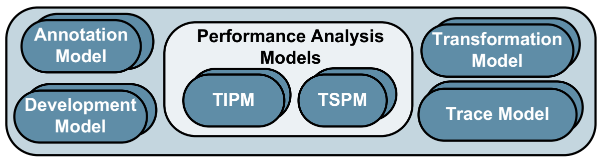


Figure 2. Models used in MDPE

Models. The latter models focus on aspects of a system relevant for performance analysis, such as behaviour information, available resources, and consumption of resources, etc.

As shown in Figure 1, MDPE uses *Tool Independent Performance Models (TIPMs)* and *Tool Specific Performance Models (TSPMs)*. The clear separation of TIPMs and TSPMs allows software vendors such as SAP to be independent of specific performance analysis tools since different TSPMs can be generated from a common TIPM. An instance for such a TSPM is the AnyLogic model [10].

The TIPM has additional advantages; we are not tied to one business modelling language but can deal with different development modelling languages. Therefore, we are required to transform Development Models to TIPMs. Currently, we support BPMN models, for instance defined with the NetWeaver BPM tool [11], UML2 models (activity diagrams in our case) created with CASE tools like Topcased [12], a hierarchical modelling language used in JPASS [13] and one SAP proprietary modelling language [14].

Initially, we used the UML SPT profile [15] to record performance related information. However, this approach only works for UML models. In order to stay independent of modelling language, we use model weaving to attach performance information to the models by using separate annotation models. This approach is described in [14] and can be used regardless of modelling language. In [8], we described the need to model not only the knowledge about historical or future instances of Development Models but also modification constraints, performance requirements and performance objectives. Thus, we are able to offer real performance related decision support provided by the *Assessment Computation Actor* (see Figure 3) to domain experts, such as proposing an optimal number of resources derived from a simulation optimization [16]. Therefore, a number of different annotation models are used in the current implementation of MDPE as can be seen in Figure 3.

The information contained in the Performance Analysis Result model can be visualized as annotations on the Development Models. Thus, *Tracing Models*, as shown by Figures 2 and 3, are used to annotate the performance engineering results back to the Development Models which the domain expert understands. In [17] our approach of tracing between the UML, the TIPM and the AnyLogic performance

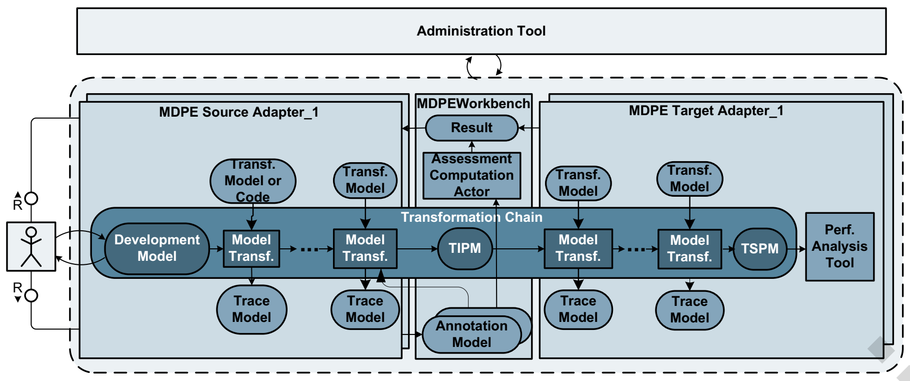


Figure 3. MDPE Transformation Chain as block diagram [5] (legend see Figure 1)

analysis tool [10] is described in detail. However, we are not only able to trace AnyLogic simulation outcomes, but all kinds of performance analysis results, as results of simulation optimizations take performance modification constraints, performance requirements and performance objectives into account.

2.2 MDPE Transformation Chain

Figure 3 shows the transformation chain that is used to transform Development Models to *TIPMs* and finally to TSPMs, such as AnyLogic models. For the definition of *Model Transformations*, we mainly used *Transformation Models* expressed in ATL [2]. In order to set up an MDPE transformation chain, transformations are needed from Development Models to the TIPM and from the TIPM to a number of TSPMs. In the current MDPE implementation, these transformations are encapsulated within source and target adapters. In more detail, the currently implemented source adapters are not transforming development models directly to the TIPM but pass through a number of transformation steps. The same is true for the currently available target adapter. This approach simplified the implementation of the transformations as it enabled us to separate different concerns in the complex model to model transformation.

For instance, the TIPM to AnyLogic transformation, which is our current TIPM to TSPM transformation, is implemented based on a two step approach: One transformation converts a TIPM into a structure of AnyLogic library objects as anticipated by AnyLogic. It generates all required objects together with additional objects required to connect everything into a working model. This structure includes all AnyLogic objects and connections between them that have to be present in the model. The second transformation applies XML formatting so that the model can be read by the AnyLogic tool. Concluding, a transformation chain from development models to TSPMs has at least a length of two: one transformation in the the source and one in the target adapter. However, in the current implementation, its length is between four to five transformations, depending on the concrete source adapter used.

2.3 Identified Problem: Administration of Numerous Modelling Artefacts

In the MDPE process we are applying more than one well-contained MDE technique at a time. Therefore, have to deal with numerous interrelated modelling artefacts to express either transformations, annotation, or tracing information. In order to implement a MDPE Workbench which is able to integrate with a number of different modelling tools, a systematic way to manage these modelling artefacts is required. In other words, metadata such as the identification and location of each model (on disk, in a repository, etc.), dependencies between these models or the semantic meaning of each of them (traceability, annotation, transformation, etc.), needs to be managed centrally.

This became especially obvious when we implemented an administration tool for our MDPE Workbench in order to manage different source and target adapters in user friendly way. This administration tool therefore needs to manage a number of different “Development Model to TIPM” transformations including the related trace models, and a number of annotation models. This implies that different transformations provided by the source or target adapter have to be dynamically added to or removed from the transformation chain. Additionally, we need to automatically trace back from e.g. performance simulation results to the development models used as input for the performance analysis [17, 14]. Hence, we are required to navigate from models in the transformation chain to their related trace models. Thus, all the involved relationships need to be managed centrally.

Furthermore, interoperability with different model repositories is required as it might happen that the modelling artefacts are distributed over a number of model repositories. In the case of SAP, proprietary modelling artefacts are processed using the SAP metadata repository called Modelling Infrastructure (MOIN) [18] and all other artefacts, such as the TIPMs and the annotation models, use the EMF [1] file-based metadata repository.

3 Megamodelling: Modelling in the Large

In the previous section we have shown that, in order to be able to deal with complex MDE environments such as in the MDPE architecture, support for uniformly managing all the involved artefacts (based on the use of their metadata) is required.

In order to provide access to this potentially huge number of MDE artefacts without increasing the accidental complexity introduced by the use of MDE, we need to identify appropriate ways to create, store, view, access, and modify metadata on these modelling artefacts. This metadata mainly represents detailed information on these involved artefacts, possibly including various kinds of links between them. This is what we refer to as *modelling in the large* [19] or *Megamodelling* [20]. In a first Subsection (3.1), the concrete need for Megamodelling is argued by identifying current common problems related to the management of MDE artefacts. In the second Subsection (3.2), a conceptual framework, which allows representing such artefacts and their metadata in order to address these problems, is proposed. Finally, in Subsection (3.3), a concrete implementation of that framework, the Eclipse-GMT AM3 project, is introduced.

3.1 The Need for Megamodelling

The models involved in complex software processes are not only UML models but can be of very various natures: XML documents that conform to specific XSD schemas or DTDs, EMF-based models that conform to different metamodels expressed in Ecore/XMI format, etc. Moreover, these models are often linked to each other and are also involved in complex chains of operations which entail, for example, model transformations. As a consequence, the solution providers need facilities (i.e. methodologies and tools) for managing all these models, specifying the possible relationships and dependencies between them and building the complex chains of operations involving all these different modelling artefacts. Thus, applying the MDE approach to a real-life use case such as MDPE means that one needs to handle a set of modelling artefacts which are:

Numerous. One application may use several hundreds of such artefacts selected from large libraries of available artefacts amounting to thousands of units or even more. As an example, within the relatively simple MDPE process, more than thirty artefacts (development/annotation/traceability models, metamodels, transformations, etc) are already involved (2.1 and 2.2).

Distributed. Some artefacts may be available on a remote site or accessible via different repositories like the MOIN and EMF ones (2.3).

Interrelated. The artefacts are related by strong semantic links. For example, a transformation refers to its source and target metamodels. These metamodels may in turn be versions or extensions of other metamodels. A model *Mb* obtained from a model *Ma* by a transformation *Mt* may record its origin model and its transformation model *Mt* (2.2). Moreover these models *Mb* and *Ma* may be related by a traceability relation (2.1). These are only a few of the semantic relations that may be found in a set of MDE artefacts.

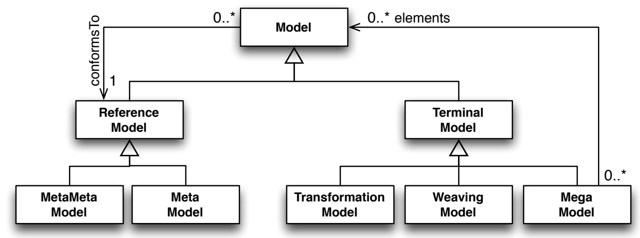
Heterogeneous. The artefacts are of different natures. They should be categorized in a systematic manner, i.e. a typing system. The simplest idea that comes to mind is to consider that all artefacts are models. Then we have a solution that consists in stating that each artefact is typed by its metamodel. This conjecture (i.e. all managed artefacts are models, conforming to a precise metamodel) has be mainly followed in the present work on MDPE (2.1).

Complex. The artefacts may contain many internal elements: for instance, a given traceability model may store several thousands of traceability links, depending on the size of the traced models (2.1). Here again, when following the simplifying conjecture stated above, the possible nature of elements contained in one artefact is defined by the metamodel.

Because of these characteristics, the (accidental) complexity of managing all the MDE artefacts in a realistic situation, such as in the MDPE process, can increase dramatically. By applying *Megamodelling*, we aim to reduce this complexity by offering a generic and extensible environment to manage MDE artefacts and their metadata at a higher level (i.e. at the model level). Searching in a large distributed library of models or chaining many different transformations are examples of *Megamodelling* operations that should be more easily performed. Thus, the proposed *Megamodelling* approach is about facilitating these operations by applying general MDE principles for model handling.

3.2 A Conceptual Framework for Megamodelling

A Megamodelling approach is based on several general concepts (Figure 4) which can be mapped to concrete cases like MDPE. Most of these concepts, corresponding to a generic conceptual MDE framework, have already been presented in [21]. In addition to these, the concept of a *megamodel* is introduced here as a building block of the *mod-*



```
context MetaMetaModel inv: self.conformsTo = self
context MetaModel inv: self.conformsTo.oclIsKindOf(MetaMetaModel)
context TerminalModel inv: self.conformsTo.oclIsKindOf(MetaModel)
```

Figure 4. Megamodelling Conceptual Framework

elling in the large approach. The principle is the following: for each real-world complex system or process, there can be a *megamodel* [22] representing the different artefacts involved (i.e. models) and their relationships by specifying associated metadata. The type of an artefact or a relationship between some artefacts, the identifier of a given artefact and its locator, etc. are examples of such registered metadata. To illustrate this, an overview of the Megamodelling conceptual framework is shown in Figure 4.

MDE approaches generally introduce three different kinds of models:

- *terminal models* (M1) which conform to *metamodels* and are representations of real-world *systems*;
- *metamodels* (M2) which conform to *metametamodels* and define domain-specific concepts;
- *metametamodels* (M3) which conform to themselves and provide generic concepts for metamodel specification.

Several kinds of *terminal models* such as *transformation models* and *weaving models*, may be considered. A *megamodel* is also a specific kind of *terminal model* whose elements represent models themselves as well as relationships between them. As it is a *terminal model*, a *megamodel* conforms to a specific *metamodel*: the *metamodel of megamodel* or *mega-metamodel*.

To summarize, a megamodel can be viewed as a metadata repository where representations of models and links between them are stored and made available to users for various and varied purposes. If represented as models, available tools, services and service parameters may also be managed by the megamodel. There are many events that may change the megamodel, like the creation or suppression of a model or a metamodel for instance. However, the advanced management of these events is not in the scope of the present work.

3.3 The Eclipse-GMT AM3 Megamodelling Solution

The Eclipse.org *AM3* solution implements the previously described conceptual framework, as an answer to the various requirements exposed in section 3.1, and thus can be used in the context of the MDPE use case. It is a project which is part of the *GMT* subproject, which is itself part of the top-level Eclipse *Modeling* project. As an Eclipse project, the AM3 prototype is fully open-source and thus all its source code is freely available from its Eclipse website and download server (*www.eclipse.org/gmt/am3/*). The generic and extensible AM3 Megamodelling solution provides not only the capabilities to explicitly specify the metadata associated with a given system or process, but also a standard *Megamodel Navigator* as well as generic and extensible editors for instantiating and editing the megamodel in a more user-friendly way. In addition, it offers several extension points allowing the definition of domain-specific extensions of the tool (i.e. extending both the metamodel of megamodel and the related UI components). Thus, AM3 is composed of two distinct sets of Eclipse plug-ins:

- The *core* plug-ins provide the basic metamodel of megamodel, the core runtime environment, the main APIs and associated generic navigator and editors.
- The *extension* plug-ins provide extensions of the metamodel of megamodel and corresponding extensions of the UI (for instance specific editor pages, contextual actions, etc).

With AM3, users can build their customized Megamodelling solution by extending either the core plug-ins or other already existing extension plug-ins. Indeed, a set of generic MDE extensions have already been developed: *GMM* for Global Model Management, *GMM4ATL* for model transformation with ATL, *GMM4CT* for Composite Transformations, etc. Some experiments with these extensions on real-life use cases have been performed, such as the ones on MDPE described in the next section.

4 Megamodelling Application on MDPE

At this point, we have introduced the MDPE case in order to motivate the need to manage all MDPE modelling artefacts in the mentioned administration tool. More specifically, we need to be able to systematically locate traceability information, specify the transformation chain and work with different model repositories.

Furthermore, we have presented an overview of megamodelling, our approach for managing the artefacts and related metadata within complex MDE environments. This section first extends the metamodel of the megamodel (presented in Subsection 4.1) so that it better supports MDPE and other similar concrete approaches. In Subsection 4.2) we further describe how a MDPE megamodel can be populated and navigated by using both standard and extended AM3 UI components. Note that, in the following, we refer to the metamodel of megamodel as *mega-metamodel*.

4.1 Extending the Mega-Metamodel

MDPE uses different types of models throughout the transformation chain. A number of Development Models and others are regular *terminal models* that can be described using the basic megamodelling concepts. Additionally, MDPE uses annotation and tracing models that express certain relationships between the different types of models. These models are produced either manually or automatically by the transformation chain.

The distinction between tracing models, transformation models and annotation models as used in MDPE was not originally supported by the mega-metamodel. Thus, in order to make megamodelling more usable for MDPE, we needed to extend the standard megamodelling concepts. The extension enables us to navigate through the different MDPE artefacts in a more transparent and specialized way. The most important concepts are summarized in Figure 5.

We have introduced new elements in three areas; from left to right in the figure: navigation across MDE artefacts, MDPE-specific model types and modelling of transformation chains. We introduced the *ModelAnnotation* and a *ModelTrace* relationships to allow high-level navigation between models related through annotation and traceability information. They are respectively associated with an *AnnotationModel* and *TraceModel*. The latter elements represent low-level annotation/trace models that relate individual model elements. The *ModelTrace* element allows us to locate related source and target models (through associations inherited from *DirectedRelationship* and *ModelWeaving*) and a *TraceModel* that contains model-element level links between the source and target. In the case of *ModelAnnotation*, the relationship is not directional. There is only one model and an *AnnotationModel* – the latter adding annotations to the former.

In Section 2, a transformation chain from development models to a TSPM via a multi-step transformation chain has been presented. We introduced the *ModelTransformation* element to represent a single transformation that contains a number of input/output *Parameter*s which are typed by *ReferenceModel*s. A *ModelTransformation* can be specified in terms of a *TransformationModel* (for ATL transformations for instance) or this link can be omitted (for transformations specified in Java, for example). At several points in the MDPE transformation chain, traceability information is produced automatically. In order to enable automatic insertion of the necessary *ModelTrace* and *TraceModel* elements into the megamodel, we introduced *TraceParameter*. The latter specifies how traceability produced by a transformation relates its output to its input. A similar approach is explained in [23].

Finally, we introduced the *TransformationLaunch* and *TransformationLaunchArgument* elements to support an automated execution of our model transformation chain. This can, for example, be accomplished by generating ANT launch configurations for the execution of the whole chain.

4.2 Tool Support for the MDPE Extension

By extending the basic mega-metamodel we are able to model the complete MDPE process as a megamodel. By loading this extension into the AM3 tool, we automatically get basic tool support to create and edit MDPE megamodels. Figure 6 depicts a screenshot of a megamodel for MDPE defined with the current version of AM3.

There are two distinct parts in the AM3 user interface. The two columns on the left hand side show the mega-metamodel structure and the elements that are contained in the megamodel. The meta-elements on the left roughly correspond to the elements shown in Figure 5. We can see several meta-elements: annotation models, tracing models, models in the transformation chain and model transformations. If we select one of these meta-elements on the left, the corresponding instances in the megamodel are shown on the right. In the figure, the terminal models in the MDPE process are shown at a given time. The right part shows a detailed overview of the currently selected model element – TIPM_1. In the figure, the standard view for a terminal model is shown. This view displays the identifier and location of the actual TIPM model and in which *Relationships* it participates (*sourceOf*, *targetOf* and *relatedTo*). If desired, we could provide a more specialized view for certain model element types. In order to do this, we need to use the UI extension point of AM3 (see subsection 3.3). In our case, a specialized view would be appropriate for visualizing and configuring the transformation chain.

5 Experiences Gained

Figure 7 summarizes how the megamodel can be used for MDPE to manage the modelling artefacts involved. All actors of the *MDPE Workbench* can query the megamodel but they still have direct access to the models. However, they can now be accessed based on their relationships to other models via the megamodelling tooling. Note that other tools can be connected to the same megamodel simply by calling the AM3 provided API.

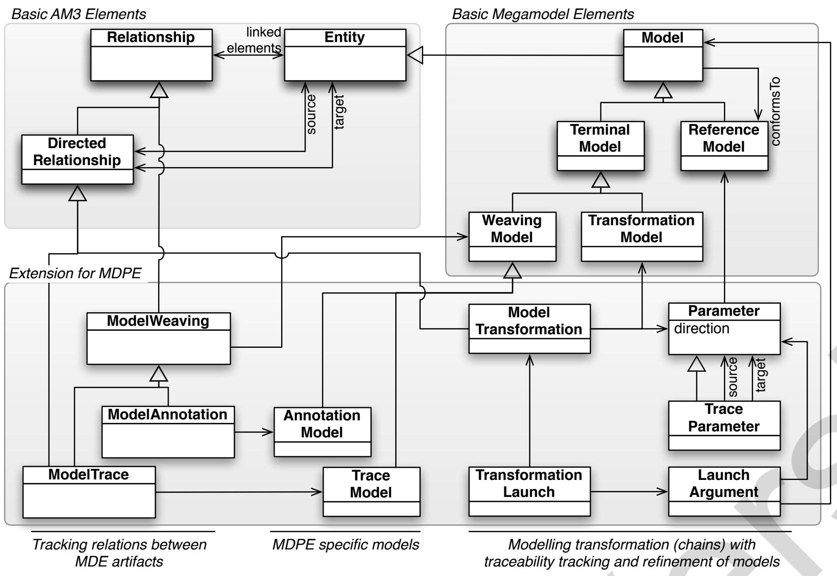


Figure 5. Extract of the MDPE Mega-Metamodel extension

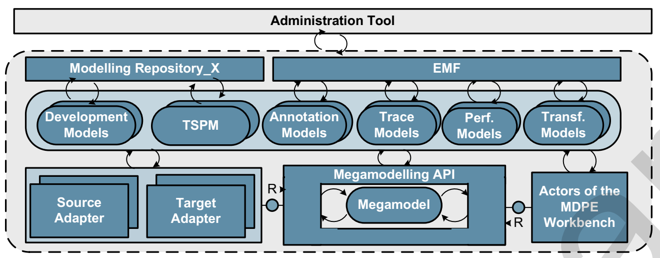
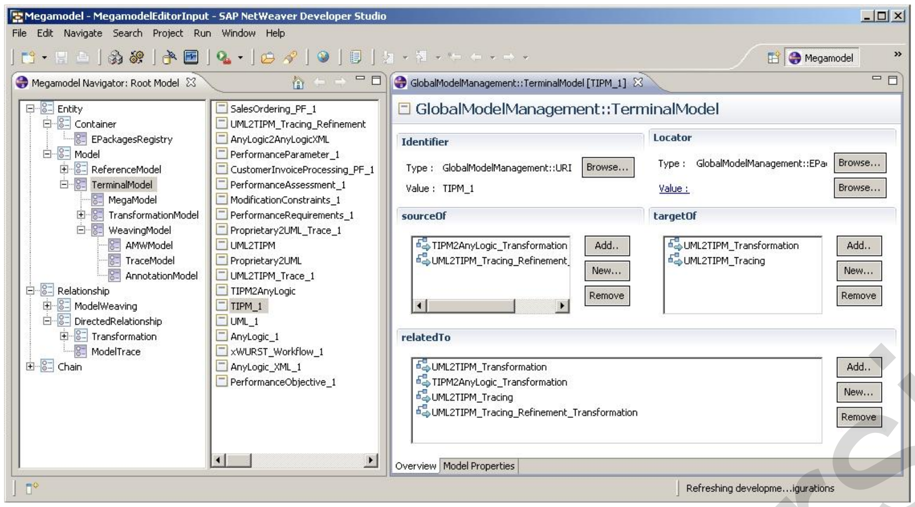


Figure 7. MDPE with mega-modelling as block diagram [5]) (legend see Figure 1)

5.1 Benefits of Applying Megamodelling

One of the main advantages gained with megamodelling is the uniform treatment of all MDE artefacts in the MDPE environment. Metadata such as the identification and location of each model (on disk, in a repository, etc.), dependencies between these models or the semantic meaning of each element (traceability, annotation, transformation, etc.), is now centrally managed in a megamodel which can be integrated into the MDPE workbench. This relieves us from the cognitive burden of keeping track of all these artefacts and their relationships manually. The MDPE megamodel has made it much easier to locate a model, a metamodel or a given transformation. It also helps to change the configuration of the transformation chain and navigate across the generated traceability links. This can be done with the specific administration we described in Subsection 2.2. As formerly described, each source and target adapter, selected with our adminstration tool, delivers a number of transformations. Applying the megamodelling approach, the administration tool is now able to automatically register the delivered transformations into the megamodel. This corresponding metadata can be used, for instance, to automatically generate ANT scripts for the launching of the MDPE transformation chain. If a specific MDPE transformation chain is executed, the MDPE Workbench updates the megamodel with the input development model, the annotation models and the generated in-between models and trace models of the chain. Thus, all metadata on the MDE artefacts of the MDPE case are now explicitly represented and stored as a model itself (i.e. a megamodel) and conform to a well-defined metamodel (i.e. an extension of the metamodel of megamodel). This means that any MDE operation such as model transformation, model weaving, model comparison, model merging can also be applied to the megamodel (and thus to the stored metadata).

By combining various MDE techniques, it is now much easier to use, analyze, process or transform the available metadata for many different purposes. Thus, we are now able, as shown by Figure 8, to use our megamodel and the generated trace models (produced by the executed transformation chain) as source models of a transformation to generate one trace model which directly links TSPM and Development Model elements. This can be done straight forward as the megamodel allows backward navigation throught the exectuted transformation chain: models of the transformation chain are related with the trace model via the megamodel. The same principle can be applied for annotation models which are associated to the development model and used in later steps in the model transformation chain.


Megamodel - MegamodelEditorInput - SAP NetWeaver Developer Studio
File Edit Navigate Search Project Run Window Help
Megamodel
Megamodel Navigator: Root Model
GlobalModelManagement::TerminalModel [TIPM_1]
Overview
Model Properties
Refreshing developme...igurations

Entity
Container
EPackagesRegistry
Model
ReferenceModel
TerminalModel
MegaModel
TransformationModel
WeavingModel
AMWModel
TraceModel
AnnotationModel
Relationship
ModelWeaving
DirectedRelationship
Transformation
ModelTrace
Chain

SalesOrdering_PF_1
UML2TIPM_Tracing_Refinement
AnyLogic2AnyLogicXML
PerformanceParameter_1
CustomerInvoiceProcessing_PF_1
PerformanceAssessment_1
ModificationConstraints_1
PerformanceRequirements_1
Proprietary2UML_Trace_1
UML2TIPM
Proprietary2UML
UML2TIPM_Trace_1
TIPM2AnyLogic
TIPM_1
UML_1
AnyLogic_1
xWURST_Workflow_1
AnyLogic_XML_1
PerformanceObjective_1

GlobalModelManagement::TerminalModel

Identifier
Type : GlobalModelManagement::URI
Value : TIPM_1

Locator
Type : GlobalModelManagement::EPa
Value :

sourceOf
TIPM2AnyLogic_Transformation
UML2TIPM_Tracing_Refinement

targetOf
UML2TIPM_Transformation
UML2TIPM_Tracing

relatedTo
UML2TIPM_Transformation
TIPM2AnyLogic_Transformation
UML2TIPM_Tracing
UML2TIPM_Tracing_Refinement_Transformation

Figure 6. MDPE Mega-Model in the current Mega-Modelling Prototype

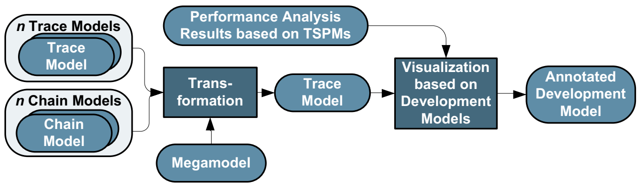

n Trace Models
Trace Model
n Chain Models
Chain Model
Performance Analysis Results based on TSPMs
Trans-formation
Trace Model
Visualization based on Development Models
Annotated Development Model
Megamodel

Figure 8. Example usage of the megamodel as block diagram [5]) (legend see Figure 1)

Concluding, the megamodel is used to systematically navigate between models and associated relationships. It is therefore no longer required to manually specify the relationship between models in the transformation chain for each different tracing application. We may also build different views on the same metadata based on various view points, for instance, to support different configurations of the MDPE process. We may then provide user-friendly representations of the configurations in several formats: SVG graphical representations of the involved transformation chains can already be generated but other formats may also be considered in the future (e.g. HTML, a word processor or spreadsheet format, etc). Having all this metadata available as a model means that we can reuse this data for many different purposes.

5.2 Identified Problems and Possible Solutions

Even though the megamodelling approach offers many benefits, not all problems are automatically solved by the current implementation of our prototype. Issues may be encountered when trying to use some of the described MDPE artefacts in certain MDE operations. Indeed, the MDPE case uses two different model repositories (MOIN and EMF) and corresponding metametamodels (MOF and Ecore), with metamodels and related terminal models, belonging to both of them. Both kinds of models cannot be used transparently as inputs for any kind of transformation because the ATL implementation currently does not support MOIN models. For the current implementation the first transformation in the chain has been defined using Java. The transformation generates not only an in-between model in the transformation chain, but also the required trace models. This use of Java enables us to bridge the two repositories as it outputs EMF models.

Another solution would have been to introduce a MOF(MOIN)-to-Ecore(EMF) conversion at some points in the transformation chain. However, this would have broken the trace links between real input MOIN models and the generated output EMF models. As a consequence, offering better support for combining heterogeneous repositories and metametamodels seems to be an important subject of further investigation. A first step in that direction will be to natively include some standard converters (for instance MOF-to-Ecore and Ecore-to-MOF ones) in the megamodelling infrastructure.

Apart from the MDPE case study, the megamodelling approach has already been applied to other smaller MDE case studies. In many of these situations the same kinds of elements have been encountered: transformation chains, traceability models, annotation models, etc. Therefore, it seems beneficial to factor out many of these elements to a higher level and to create a more elaborate MDE megamodelling framework extension, and basically provide corresponding UI features. This is what has been performed on the *GMM* extension (cf. section 3.3) which has been upgraded accordingly, by applying this principle, within the context of our experiments on the MDPE use case. Thus, new extensions of this generic framework (i.e. of this GMM extension) can then leverage the out-of-the-box support, which lowers the effort to extend the mega-metamodel for a more specific MDE domain. Moreover, this will ensure better compatibility between different extensions and will also reduce the time invested in the development of additional UI components.

6 Related Work

Due to the numerous modelling artefacts in the MDPE process conforming to numerous metamodels, we faced a set of problems, regarding the management of these artefacts, which were not relevant so far for the related work mentioned above.

Most so-called Integrated Development Environments (IDEs) use some kind of internal information model to record relationships between a limited number of predetermined artefact types such as source code files, configuration files and, increasingly, models. For many projects we must use multiple tools, and hence multiple types of artefacts, to support different concerns. This means that we cannot rely solely on the internal information model of a single tool.

The discipline of managing and interrelating different kinds of (software) artefacts in general is often referred to as software asset/artefact management or even more general metadata management. In [24], a large case study, involving many different kinds of business-oriented models and other artefacts is presented. The artefacts involved have similar characteristics as we discussed in Section 3. Different kinds of techniques to manage these artefacts were tried out, amongst them a dedicated asset management tool (Model Blue). The purpose of the latter is similar to AM3, although it is not model-based. Another field of similar research is the traceability domain. Often, we want to trace artefacts across subsequent refinement steps: from requirements via design models to code. An example of such an approach is proposed in [25]. The mentioned works in the fields of asset traceability and management support a predetermined set of artefacts. With AM3 we try to provide a generic infrastructure that can be extended to manage any kind of modelling artefact.

Other approaches that aim to offer a generic model management infrastructure are proposed in [26] and [27]. The first focuses managing data models such as Entity Relationship, SQL and XML. The latter builds on the ideas of the former, but aims to support any kind of model. Both emphasize the need to standardize model operations such as matching, merging and composing. In AM3, we emphasize information structure and do not aim to standardize model operations; these should be provided by its extensions. In [28], we have developed a model-based approach to define and execute transformation chains. Transformation chain models contain elements that represent MDE artefacts such as models, metamodels, traceability models and transformations. This can hence be considered as a specialized megamodelling approach.

7 Conclusions and Future Work

Model Driven Performance Engineering (MDPE) involves many different artefacts such as models (in the broad sense: i.e. terminal models, metamodels and metametamodels) and chains of model transformations, all taking part in a complex and integrated MDE approach. As their number increases, the need for a management infrastructure dedicated to the associated metadata becomes more and more important.

In this paper, we presented the *megamodelling* approach and showed how it is aimed at dealing with a large amount of MDE artefacts of numerous kinds. MDPE was then used as an industrial real-life MDE scenario in order to illustrate that the described approach can bring many benefits to the management of such an MDE infrastructure. By extending our basic megamodelling framework with the required concepts, we are now able to support uniform management of the numerous, distributed, heterogeneous and complex MDE artefacts that are involved in the MDPE case study. A lot of the accidental complexity which is related to the manual management of these artefacts is thus hidden behind the Eclipse-GMT AM3 tool interface. As explained in Section 5, the results obtained from our initial experiments are very satisfying. However, we also came to the conclusion that, in many MDE scenarios, we will need to manage similar artefacts as in the MDPE case. Therefore, in our future work on the AM3 tool, we will mainly focus on designing a more elaborate base mega-metamodel, the corresponding extended UI, and basic generic support for executing all kinds of transformation chains.

Acknowledgments. The information in this document is proprietary to the following MODELPLEX [29] consortium members: SAP RESEARCH and INRIA. The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law. COPYRIGHT 2008 BY SAP RESEARCH AND INRIA

References

[1] Budinsky, F., Steinberg, D., Ellersick, R., Merks, E., Brodsky, S.A., Grose, T.J.: Eclipse Modeling Framework. Addison Wesley (2003)

[2] ATL project: http://www.eclipse.org/m2m/atl/ (2006)

[3] OMG: Mof2.0 query/view/transformation (qvt) adopted specification. omg document ptc/05-11-01, available from www.omg.org, OMG (2005) 193–204

[4] Didonet Del Fabro, M., Bézivin, J., Jouault, F., Breton, E., Gueltas, G.: Amw: a generic model weaver. In: 1re Journe sur l'Ingnierie Dirige par les Modles (IDM05). (2005)

[5] Knöpfel, A., Gröne, B., Tabeling, P.: Fundamental Modeling Concepts: Effective Communication of IT Systems. John Wiley & Sons (2006)

[6] Fritzsche, M., Gilani, W., Fritzsche, C., Spence, I., Kilpatrick, P., Brown, J.: Towards utilizing model-driven engineering of composite applications for business performance analysis. In: 4th ECMDA-FA, LNCS 5095. (2008)

[7] Smith, C.: Performance Solutions: A Practical Guide To Creating Responsive, Scalable Software. Addison-Wesley (2002)

[8] Fritzsche, M., Gilani, W., Spence, I., Brown, T.J., Kilpatrick, P., Bashroush, R.: Towards performance related decision support for model driven engineering of enterprise soa applications. In: 15th ECBS'08, IEEE (2008) 57–65

[9] OMG: Business Process Modeling Notation Specification, Final Adopted Specification (2006)

[10] XJ Technologies: AnyLogic — multi-paradigm simulation software (2008) URL http://www.xjtek.com/anylogic/.

[11] Snabe, J.H., Rosenberg, A., Mller, C., Scavillo, M.: Business process management: The sap roadmap, SAP PRESS (2008)

[12] Pontisso, N., Chemouil, D.: Topcased combining formal methods with model-driven engineering. In: ASE '06, IEEE (2006)

[13] JCOM: http://www.jcom1.com/cms/jpass.html (2008)

[14] Fritzsche, M., Johannes, J., Assmann, U., Mitschke, S., Gilani, W., Spence, I., Brown, J., Kilpatrick, P.: Systematic usage of embedded modelling languages in model transformation chains. In: SLE 2008 (to appear in LNCS), Springer (2008)

[15] OMG: UML profile for schedulability, performance, and time specification (2005) URL http://www.omg.org/docs/formal/03-09-01.pdf.

[16] Olafsson, S., Kim, J.: Simulation optimization. In: WSC'02. (2002)

[17] Fritzsche, M., Johannes, J., Zschaler, S., Zherebtsov, A., Terekhov, A.: Application of tracing techniques in model-driven performance engineering. In: ECMDA-FA 4th workshop on traceability. (2008)

[18] Altenhofen, M., Hettel, T., Kusterer, S.: Ocl support in an industrial environment (2007)

[19] Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P.: Modeling in the large and modeling in the small. In: MDAFA 2003, MDAFA 2004, LNCS 3599. (2003)

[20] Bézivin, J., Jouault, F., Valduriez, P.: On the need for megamodels. In: GPCE at 19th OOPSLA. (2004)

[21] Bézivin, J., Jouault, F.: KM3: a DSL for Metamodel Specification. In: 8th IFIP. (2006) 171–185

[22] Barbero, M., Jouault, F., Bézivin, J.: Model driven management of complex systems: Implementing the macroscope's vision. In: 15th ECBS'08, IEEE (2008)

[23] Pilgrim, J.V., Vanhooff, B., Schulz-Gerlach, I., Berbers, Y.: Constructing and visualizing transformation chains. In: 4th ECMDA-FA 2008, LNCS 5095, Berlin, Germany, Springer (2008)

[24] Zhu, J., Tian, Z., et all, T.L.: Model-driven business process integration and management: a case study with the bank sinopac regional service platform. IBM J. Res. Dev. **48** (2004) 649–669

[25] Olsson, T., Grundy, J.: (Supporting traceability and inconsistency management between software artefacts)

[26] Melnik, S., Rahm, E., Bernstein, P.A.: Rondo: a programming platform for generic model management. In: SIGMOD'03, ACM (2003) 193–204

[27] Salay, R., Chechik, M., et all, S.E.: An eclipse-based tool framework for software model management. In: eclipse '07, ACM (2007) 55–59

[28] Vanhooff, B., Ayed, D., Baelen, S.V., Joosen, W., Berbers, Y.: Uniti: A unified transformation infrastructure. In Engels, G., Opdyke, B., Schmidt, D.C., Weil, F., eds.: MoDELS. (2007) 31–45

[29] MODELPLEX project: https://www.modelplex-ist.org/ (2006-2010)