Systematic Usage of Embedded Modelling Languages in automated Model Transformation Chains

Mathias Fritzsche[1,3], Jendrik Johannes[2], Uwe Aßmann[2], Simon Mitschke[1,2], Wasif Gilani[1], Ivor Spence[3], John Brown[3], and Peter Kilpatrick[3]

[1] SAP Research CEC Belfast
Shore Road, BT370QB Newtownabbey
United Kingdom
`mathias.fritzsche@sap.com, simon.mitschke@sap.com, wasif.gilani@sap.com`

[2] Technische Universität Dresden
D-01062, Dresden, Germany
`jendrik.johannes@tu-dresden.de, uwe.assmann@tu-dresden.de`

[3] Queen's University Belfast
University Road, Belfast BT7 1NN
United Kingdom
`i.spence@qub.ac.uk, tj.Brown@qub.ac.uk, p.kilpatrick@qub.ac.uk`

Abstract. Annotation of programs using embedded Domain-Specific Languages (embedded DSLs), such as the program annotation facility for the Java programming language, is a well-known practice in computer science. In this paper we argue for and propose a specialized approach for the usage of embedded Domain-Specific Modelling Languages (embedded DSMLs) in Model-Driven Engineering (MDE) processes that in particular supports automated many-step model transformation chains. It can happen that information defined at some point, using an embedded DSML, is not required in the next immediate transformation step, but in a later one. We propose a new approach of model annotation enabling flexible many-step transformation chains. The approach utilizes a combination of embedded DSMLs, trace models and a megamodel. We demonstrate our approach based on an example MDE process and an industrial case study.

1 Introduction

The usage of Domain-Specific Modelling Languages (DSMLs) in Model-Driven Engineering (MDE) is increasing in popularity. Traditionally, it is a common practice to define and use Domain-Specific Languages (DSLs) by embedding them into other languages [1]. Bravenboer et al [2] proposed a GUI definition language embedded into Java. Another common example is the usage of SQL statements as Strings in Java or PHP, which is effectively language embedding. While this is in general not a new idea, new challenges emerge when this practice is applied for DSMLs in the MDE domain.

The distinction between DSMLs and traditional DSLs is not formally defined. There are, however, certain aspects which tend to be different. For example, the metalanguages used to define language syntax tend to be different; DSMLs are often defined through MOF-like [3] metamodelling languages while traditional DSLs are often defined through Extended Backus-Naur Form (EBNF)-like grammars. The syntax of DSMLs tends to be more graphical, while traditional DSLs more often come with textual syntax. These are only some examples of how DSMLs and traditional DSLs tend to differ in general.

In this paper we focus on embedded DSMLs and how semantic definition and compilation of them differs compared with traditional embedded DSLs. We identify problems that arise from these differences and propose solutions to tackle the identified problems.

The difference between DSMLs and traditional DSLs is related to the difference between programming and modelling. Programs can be directly interpreted or compiled into an executable system. Models—in the sense of MDE—on the contrary are abstract descriptions that need to be enriched with additional information to obtain an executable system. Therefore, traditional DSLs—that are special-purpose programming languages—require fully defined executable semantics to be usable. DSMLs on the contrary can well be used without having full executable semantics defined. There are different approaches to define the semantics for a traditional embedded DSL:

1. **Interpreter written in the host language:** It is a common practice, for instance, to define SQL statements as Strings in Java. Those are then preserved during compilation and interpreted at runtime by a Java method. The semantics are then defined inside the host language that embeds the DSL.
2. **Compiling to the host language:** Another approach is to embed a DSL into a host language by extending the host language with new syntax (e.g., [2]). The new constructs are then compiled down to constructs of the original language.
3. **Compiling embedded and host language to a common base language:** Quite similar is the approach to extend the host language, but then compile both host and embedded into a common base language. In AspectJ for instance Java classes and AspectJ aspects are both compiled to and integrated in Java byte code.

If we look at embedding DSMLs in modelling languages in a first and using the embedded information in a second step, compilation steps might be involved—which are called *model transformations*. The most prominent example is the application of a profile in UML. A UML profile defines a DSML that can be injected into UML in a standard way. The information defined in the profile application (that is in the embedded DSML) is then interpreted in a model transformation. This transformation can also produce another UML model (e.g., Platform Independent Models (PIM) to Platform Specific Models (PSM) in MDA [4]) that resolves the information in the profile into ordinary UML elements. This is in fact a compilation to the host language (see 2 above). The transformation can also produce instances of another modelling language which is similar to compiling host and embedded language into a common base language (3 above).

Consider now the case that the result of the compilation is, after further enrichment, again automatically compiled (i.e., transformed) and so on. Such an automated many-step transformation chain is more systematic than a single transformation, e.g. to realize modularity or, in other words, to achieve separation of concerns [5] for complex transformations. In such an automated many-step transformation chain, it might well happen that information defined in an embedded DSML at some point is not required in the next compilation step, but in a later one. Since the information is required at some point, it needs to be preserved throughout the transformation chain. This has the drawback that the information that is not of interest in a particular intermediate model has to be put directly into the model anyway to be picked-up by the next transformation.

This is, however, not desired in MDE, because the target languages of transformation are actually DSMLsshould only provide constructs that are of interest for the particular domain they are specified for.

We identified this problem in a concrete MDE process called Model-Driven Performance Engineering (MDPE) [6]. The process defines an automated many-step transformation chain and requires the *annotation* of additional information at certain points in the process that is only evaluated by a transformation later in the chain and not of interest in intermediate models. As we will see, the languages in which the annotations are defined are in fact embedded DSMLs, where the language of the annotated models is the host DSML. Profiling in UML, for instance, is a specific type of annotation where UML is the host language and the annotation language (i.e., the profile) is the embedded DSML.

In this paper we propose a methodology that traces any information defined in any embedded DSML in a many-step transformation chain in such a way that it can be accessed at any point in the chain when it is needed. The approach can be applied for automated transformation chains where no human interaction is required. For this, we unify the way in which DSMLs for annotations are defined. Using this methodology, transformations and intermediate models do not need to be polluted by handling locally unnecessary metadata.

The rest of the paper is structured as follows: Section 2.2 introduces MDPE as an example of a process and a use case applying that process. Section 3 discusses the problems that occur in the application of the MDPE process with respect to handling annotations defined in embedded DSLs. In Section 4 we propose a system architecture based on MDE technologies that systematises the handling of annotations and unifies the definition and usage of embedded DSMLs for annotations. Section 5 gives details of our implementation of the architecture and discusses its advantages based on experience gained so far. In Section 6 we give pointers to related work and conclude.

2 Scenario: Language Engineering in Model-Driven Performance Engineering

In this section we introduce Model-Driven Performance Engineering (MDPE) as one example of MDE process implementing an automated many-step transformation chain.

MDPE has been defined in order to enable performance engineering based on development models[4] and additional performance related data [6]. The process therefore provides performance-related[5] decision support to business domain experts [7, 8], i.e. support decisions for resource scheduling problems via business simulations.

In the following subsection we give an overview of the industrial case study that we employed to gain experiences with the MDPE process. A brief overview of the stepwise process follows in subsection 2.2.

2.1 Use Case

In our special case, a business domain expert uses SAP NetWeaver Business Process Management (aka SAP NetWeaver BPM) to define business process extensions by applying MDE concepts. A Business Process Modelling Notation [9] based tool called Galaxy, announced in May 2008, is intended to be used for this purpose[10]. We assume the Business Process Modelling Notation as an example for a DSML.

In [11], we presented a SAP NetWeaver BPM based case study called **W**ine **U**nde**R** **S**pecial **T**reatment (xWURST). There, an already running back-end business process called "Sales Order Processing" is extended with a so-called front-end process [11] to meet the specific needs of a wine seller. The behaviour of both, the back-end as well as the front-end process, is assumed to be available as a model. For the back-end processes, SAP proprietary models are used whereas front-end processes are modelled in BPMN using the previously mentioned SAP NetWeaver BPM (see Figure 1). We used this case study to gain firsthand experience with the approach described in this paper.

Within the case study, the back-end process is extended so that it is possible to add as compensation a free bottle of wine only to orders of those customers who notified a quality issue for their previous order. The decision about which wine will be added for free has to be taken manually by a wine specialist based on the wine rating and the customer's purchase history over the last 12 months. Additionally, the selection has to be approved by a manager.

We claim that the business domain expert needs support from the business performance perspective, since there are several steps within the business applications which have to be processed by human resources. The decision about

[4] We use the term *development model* to distinguish between models as development artefacts and formal *performance models*.

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

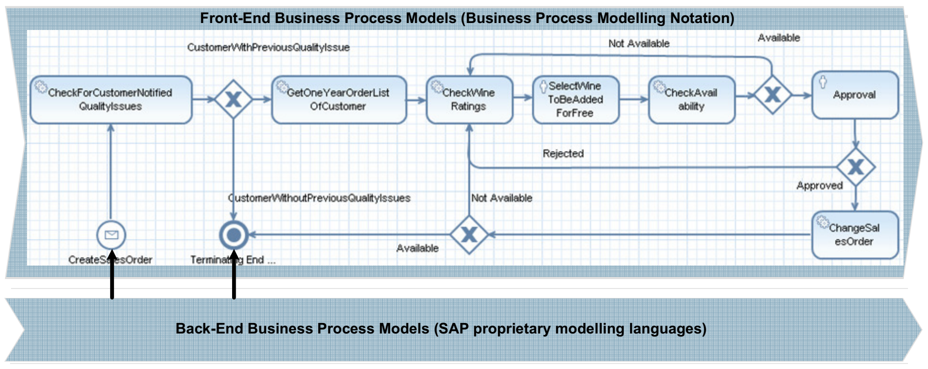

Front-End Business Process Models (Business Process Modelling Notation)
CustomerWithPreviousQualityIssue
Not Available
Available
CheckForCustomerNotified QualityIssues
GetOneYearOrderList OfCustomer
CheckWine Ratings
SelectWine ToBeAdded ForFree
CheckAvail ability
Approval
Rejected
Approved
CustomerWithoutPreviousQualityIssues
Not Available
ChangeSal esOrder
Available
CreateSalesOrder
Terminating End ...
Back-End Business Process Models (SAP proprietary modelling languages)

Fig. 1. Extension of an already existing back-end business process modelled with the Business Process Modelling Notation (BPMN)

which human resource performs which step in a business process is not trivial to make. In addition, the former difficulty is further increased by the flexibility introduced by the possibility of extending the back-end processes with front-end processes by a domain expert, with no performance related skills. This can dramatically change the behaviour of already running and stable processes.

Therefore, the necessity of providing performance related decision support for a domain expert, i.e. a business domain expert, is obvious. Such support can be provided through an MDE process that involves multiple modelling languages—including embedded DSMLs for annotations—and that is realised by various transformation, annotation, and tracing tools. MDPE is such a process, and is described in the following subsection.

2.2 Stepwise transformation for performance engineering

With the MDPE process [6, 11], we implemented an extensible approach which can be applied to different use cases, not only to the one presented in the previous subsection. A brief description of the latest version of the MDPE process is given in order to provide an example for the need of a systematic approach for model annotation with embedded DSMLs in transformation chains.

Most parts of the MDPE process are implemented within the *MDPE Workbench* (see figure 2). This workbench is developed based on Eclipse and the Eclipse Modelling Framework (EMF) in order to provide a widely usable framework. However, we developed a *MDPE Workbench Adapter4SAP* in order to use the MDPE Workbench in integration with the SAP NetWeaver BPM tooling, which is mainly based on SAP's proprietary Modelling Infrastructure (MOIN) [12].

Within the MDPE process, we do not directly transform *development models*, such as the model shown in figure 1, into *Tool Specific Performance Models*

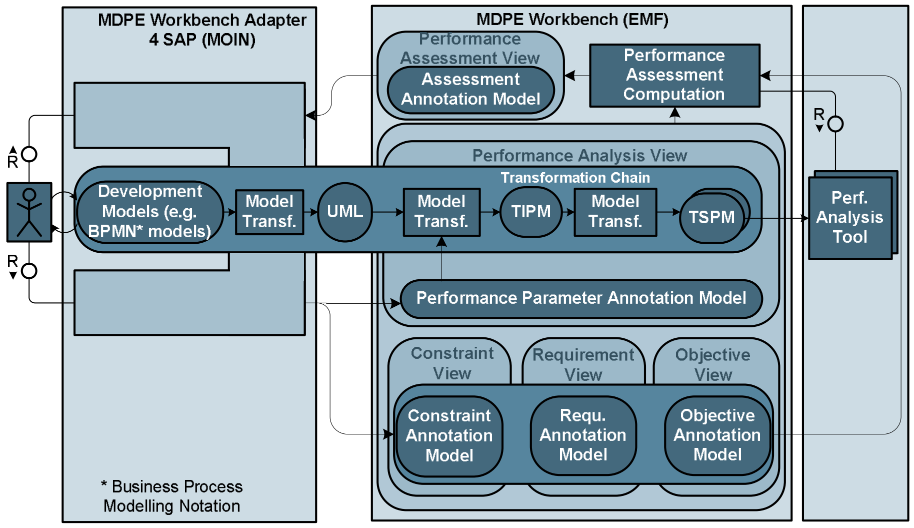

MDPE Workbench Adapter 4 SAP (MOIN)
MDPE Workbench (EMF)
Performance Assessment View
Assessment Annotation Model
Performance Assessment Computation
Performance Analysis View
Transformation Chain
Development Models (e.g. BPMN* models)
Model Transf.
UML
Model Transf.
TIPM
Model Transf.
TSPM
Perf. Analysis Tool
Performance Parameter Annotation Model
Constraint View
Requirement View
Objective View
Constraint Annotation Model
Requ. Annotation Model
Objective Annotation Model
R
R
R
* Business Process Modelling Notation

Fig. 2. Model-Driven Performance Engineering as block diagram ([14])

(TSPMs) but stepwise via intermediary *Tool Independent Performance Models (TIPMs)*. Thus, we decided to implement our transformation in a modular way. This enables us to deal not only with other proprietary modelling languages, such as the languages for the purpose of back-end process modelling, but also well-known modelling languages, such as UML. Additionally, tool independence is of high value for business software vendors, such as SAP, in order to be independent of one specific simulation tool, such as the one employed in the case study called AnyLogic [13]. The TIPM metamodel represents behaviour information, available resources and consumption of resources.

However, the current implementation of the so-called *MDPE Workbench Adapter4SAP* is not transforming proprietary models directly to the TIPM but to UML models in order to build a bridge between the TIPM and SAP proprietary models. We use this approach as it enables SAP to share proprietary models with partners of research projects without publishing the metamodels. Additionally, we are able to modify the content of the proprietary models slightly within a transformation to UML models in order to avoid publishing confidential details of the implemented business processes.

It is obvious that we are required to annotate *performance parameters* (see Figure 2), such as resource demands, branch probabilities, and factors due to contention for resources in order to transform our development models stepwise so as to prepare them for performance simulation.

In the use case, introduced in the previous subsection, we are required to annotate the average resource demands. Annotations have to be done in terms of employee time, for each manually processed step and the resource mapping, in particular, how many employees are working on which manual step in our behaviour models. This data can be extracted from the past history if the process has already been productive, and assumed values have to be annotated for newly defined parts of a business process.

In [7] we described the need to model not only the performance parameters, but also *Modification Constraints*, *Performance Requirements* and *Performance Objectives* in order to provide performance related decision support for business domain experts who generally lack performance expertise. Therefore, we additionally need to annotate development models with this kind of information, as shown in Figure 2. The information is used in order to compute the *Performance Assessment Computation* view to be presented to the domain expert. Based on the annotated information, the Performance Assessment View can, for instance, contain a proposal guiding a business domain expert to the optimal resource mapping or optimized design solution.

In [8] we described the tracing of simple simulation results back to the development models by using *Trace Models*, storing the information about which source element(s) is mapped to which target element(s) via a model transformation. We use the same approach in order to trace the more beneficial Performance Assessment View.

The additional step in the chain (transforming SAP models to UML), but also the modelling of Modification Constraints, Performance Requirements and Performance Objectives, which was not considered in [8], introduces new challenges with respect to annotation and tracing of the annotated information in the chain. The following section discusses these challenges.

3 Problem Motivation

It is obvious that we are required to annotate additional data, such as performance parameters (cf. Figure 2) to the business process models (cf. Figure 1), as described in the previous section.

The interesting point about the annotated information is that it is accessed only after a number of automated transformation steps. It can be seen in Figure 2 that we use the Performance Parameter information initially in the second transformation step to generate a TIPM. The annotated Modification Constraints, Performance Requirements, and Performance Objectives are initially used for the Assessment Computation (cf. upper part of Figure 2) after the TIPM has been generated. However, all annotations are performed based on the development models as the business process modeller does not wish to access models which are pure performance analysis models, such as the TIPM. For our initial UML-based implementation of a preliminary version of the transformation chain we used UML Profiles as a UML embedded DSML for the annotation of the Modification Constraints, Performance Requirements and Performance Objectives, as they are well supported by UML-based tools [7].

However, we identified a number of shortcomings with this approach when we started to work with a longer transformation chain and with modelling languages other than UML.

Non-UML modelling languages, such as models conforming to the Business Process Modelling Notation and SAP proprietary languages used for our current case study (see subsection 2.1), do not normally support a profiling mechanism which is an extension mechanism on the meta-level.

However, modifying the metamodel of proprietary models is not a systematic solution as it would require polluting the host DSML with information that is not domain-specific. This turns out to be a problem especially if models are provided by a third party. This is true for back-end business processes in our case study (cf. Subsection 2.1), which may partly be provided by SAP and partly by a third party. In such cases we might not be able to extend the modelling language of the third party process, and, therefore, cannot annotate the third party models if they do not provide any extension mechanism.

In addition, transformations that do not require access to the annotated information, such as the transformation from development models to UML (cf. left side of Figure 2), obviously should not be polluted with the annotated information, since the annotated information is only needed in the UML to TIPM transformation. Alternatively, it would be possible to use the originally extended development model as an additional input for the UML to TIPM transformation. However, both options are not effective as in both cases more data is used as an input for a transformation than required. Hence, metamodel extensions, such as provided by UML profiles, do not support chains of transformations systematically.

Therefore, an approach that enables host model extensions without manipulating them in an MDE process where the metamodels do not support systematic model extensions and the metamodel can or should not be changed is needed here. Furthermore, the approach needs to explicitly support tracing of annotated information in transformation chains such that information can be accessed directly only when it is needed and not earlier for preservation reasons. In the next section we define such an approach.

4 Proposed Solution

Figure 3 gives an overview of our proposed architecture enabling systematic annotations for model transformation chains, namely the MDPE transformation chain in our case.

Our approach takes the following three model types into account in order to annotate development models (cf. left side of Figure 3):

1. **Annotation Models**, as described in [15, 16], are instance of a specific isolated metamodel. Such a metamodel effectively defines an embedded DSML and consists of two parts: one for the annotation itself, and one for the information about how annotations are composed with the original host models, such as the development models in the MDPE case. The first (annotation) part defines the domain of the embedded DSML—for example, performance parameters or performance objectives. The second (weaving) part, defines

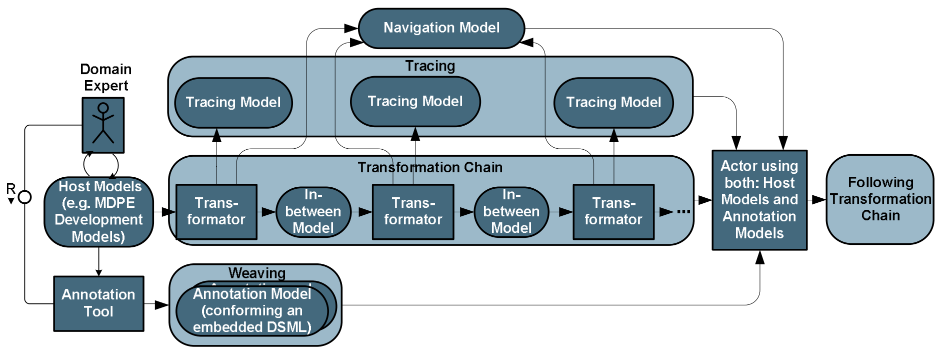


Fig. 3. Proposed Architecture

how the language is embedded into the host language—that is, which annotations are linked to which elements in the development models. Due to the fact that this linkage information is included in the Annotation Model, it is not necessary to pollute the original development models with this data or to extend their metamodel(s).

2. **Tracing Models** as described in [17–19] contain information about which source model elements have been transformed to which target model elements in a transformation chain (cf. centre of Figure 3). Hence, tracing models contain associations between models conforming to two different metamodels. We use tracing models in our approach as input for tools which are actually using the annotated information. The trace model is required in order to know at any position p in a model transformation chain, which model element(s) el_p are associated with which source model element(s) el_{source}. This is necessary in order to know which annotation is (indirectly) linked with which model element el_p.
3. A **Navigation Model** is used to associate models in a transformation chain with the related tracing models, such as described in [20]. This is required since we assume that the models in a transformation chain are non-changeable and therefore cannot be polluted with the tracing model linkage information.

5 Implementation

A description of an implementation of the concepts mentioned in the previous section is provided in this section.

5.1 Annotation Models

The annotation of models in transformation chains, such as the development models shown in Figure 1, is done based on Annotation Models as briefed in the previous section.

We have developed a generic approach which is, for instance, usable for the annotation of development models in the MDPE process. For this process, we are required to support a number of different embedded DSMLs used at different steps in the MDPE transformation chain as shown in Section 2.

It can be seen in the upper part of Figure 4 that our approach inherits from the weaving metamodel *MWCore* provided by the ATLAS group [21]. We selected this metamodel due to its integration in the Eclipse framework.

In comparison to other model annotation approaches based on the MWCore metamodel ([15] and [16]), our annotation approach is based on a generic metamodel for model annotations, the *Annotation Meta Model* (AMM, see middle part of Figure 4). It refines the meta-class *WLink* from the MWCore metamodel with the meta-class *Annotation.* The WLink class is indirectly associated with an attribute "ref" of type String that is used to represent references into our proprietary modelling repository MOIN. Hence, the Annotation model elements represent the weaving link to the host DSML elements, which can be defined in any kind of modelling repository, which includes the SAP proprietary MOIN repository for our MDPE case. Additionally, an Annotation contains references to *AnnotationProperties* which represent the annotated information to the source model elements.

An AnnotationProperty can be further refined for the specific needs of a specific annotation metamodel (that is a specific embedded DSML). In order to well integrate our work in the Eclipse workbench, the extension of the Annotation Meta Model can be done by implementing an Eclipse Extension-Point which is provided by the plug-in implementing the Annotation Meta Model.

This architecture of loose coupling is employed by our generic editor for model annotations in order to support navigation through a specific annotation metamodel. Thus, based on the Annotation Meta Model our generic editor is usable for all kinds of metamodels which are extending the Annotation Meta Model, and is integrated in the Eclipse Workbench.

Additionally, the generic editor currently allows annotation of all kinds of AnnotationProperties to all kinds of model elements in the MDPE development models. In future, we anticipate to configure this mapping in a separate constraint model. However, in our current implementation we have hard coded the required constraints.

Additionally, we are able (through an extension point) to refine some classes of our generic editor in order to make it even more suitable for users. Figure 5 depicts the user interface of our editor for the Business Performance Annotation Meta Model, as one embedded DSML example.

The metamodel of this embedded DSML is depicted by the package called "BPAMM" at the lower part of Figure 4. It enables the specification of multiple business *Scenarios* in order to separate multiple cases for performance simula-

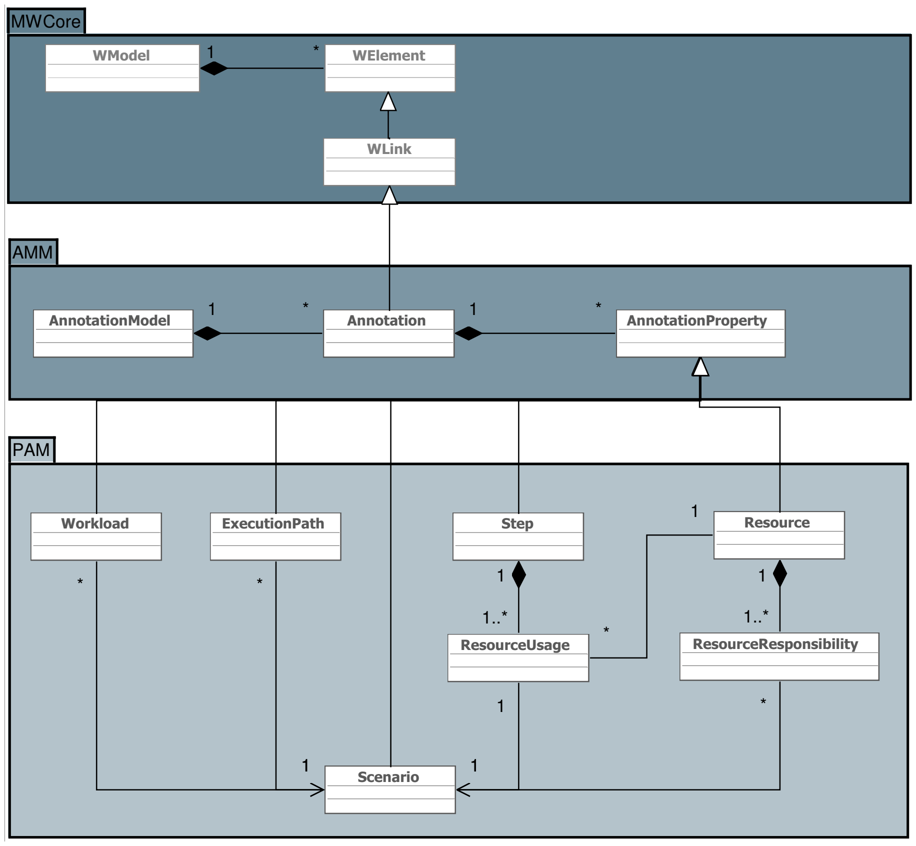
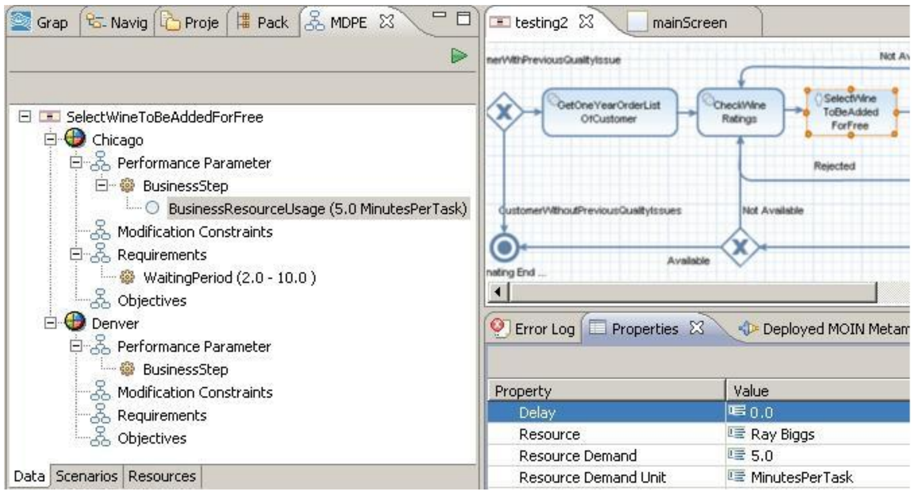


Fig. 4. Example for an embedded DSML used for the MDPE process (simplified meta-model)

tion, for instance, different locations of the Wine Seller company, introduced in subsection 2.1, such as "LocationChicago" and "LocationDenver" (see Figure 5). A business scenario is additionally related with a number of *Workloads*, *ExecutionPaths* and *ResourceUsages*.

Additionally, *Resources* can be defined which are aligned with a *ResourceResponsibility*. We use this in order to annotate a BusinessResource, such as the human resource "WineSpecialist_RayBiggs", to a development model in the MDPE transformation chain (see figure 1). For each BusinessResource, one or multiple *ResourceResponsibilities* have to be defined in order to specify operation times for different Scenarios in a business. It is, for instance, possible to specify an operation time of "8_HoursPerWorkingDay" for the "WineSpecialist_RayBiggs".

The annotation of an ExecutionPath enables annotation of a probability based on a modelled behaviour as depicted by Figure 1. In particular, it is possible to model how often a specific path in a business process is executed in



```
SelectWineToBeAddedForFree
  Chicago
    Performance Parameter
      BusinessStep
        BusinessResourceUsage (5.0 MinutesPerTask)
    Modification Constraints
    Requirements
      WaitingPeriod (2.0 - 10.0 )
    Objectives
  Denver
    Performance Parameter
      BusinessStep
    Modification Constraints
    Requirements
    Objectives
```

Property	Value
Delay	0.0
Resource	Ray Biggs
Resource Demand	5.0
Resource Demand Unit	MinutesPerTask

Fig. 5. Annotated action “SelectWineToBeAddedForFree”

case there are more than one possibilities. We used this in our case study (see subsection 2.1), for instance, to specify that 10 percent of all approvals regarding which wine is added for free are rejected by the manager of the wine specialist.

BusinessSteps are used to annotate actions in our development models with resource demands and to associate them to a Resource. As can be seen by the screenshot in Figure 5, the manual action “SelectWineToBeAddedForFree” requires 5 minutes of the resource “WineSpecialist_RayBiggs” in “Chicago”.

5.2 Tracing Models

For the tracing issue we used the tracing metamodel by the ATLAS group [18]. There are other tracing metamodels available as well, such as [17]. However, for our MDPE case we have selected this metamodel since the ATLAS group also developed the Higher-Order Transformation mechanism [22, 18] that we have employed for most transformations in our model chain. Higher-Order Transformations are transformations that are used to transform one transformation A to a new transformation $A*$. The approach enables us to generate tracing models with minimal additional effort in our MDPE process, as presented in [8]. Higher-Order Transformations can be integrated in a batch-job as described by Jouault [18] in order to automatically extend the existing transformations in a way that they additionally output tracing models. This approach is used in [18] to automatically extend rules, for instance, within ATL [23] transformations, with code for additional information generation. This code creates a tracing model when the transformation is executed. Hence, tracing is achieved with minimal additional effort via Higher-Order Transformation.

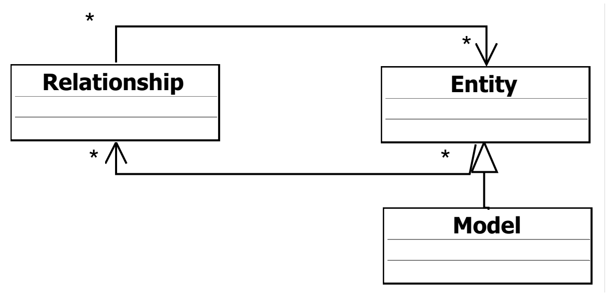


Fig. 6. Simplified metamodel of the megamodel defined in [20]

5.3 Application of a Megamodel as Navigation Model

Based on the MDPE process, we experienced the requirement to be able to navigate between the modelling artifacts by taking their interrelationships into account. In particular, navigation from models in the transformation chain to their related trace models is required in order to know at any step in a model transformation chain which trace model is related to which model in the chain in order to retrieve the corresponding annotation information.

To our knowledge, there is just one approach available fulfilling this requirement called Megamodelling [20]. A megamodel represents *Models* (see Figure 6) and their *Relationships* (see Figure 6). Our current implementation defines the megamodel [20] , and add not only the different MDPE models, i.e. the development models (see Figure 1), and the generated UML model, the TIPM, the AnyLogic Simulation model and the trace models, but also Relationships among them, to it.

5.4 Use of transformation results and annotated information

Finally, there is a step within or after a transformation chain where the annotated information is required to be computed. A description about how this computation is done, for instance, for the MDPE case, introduced in subsection 2.1, follows.

Use of an embedded DSML for the UML to TIPM transformation in the MDPE process: Due to the tracing models and the megamodel, mentioned in the previous subsections, the transformation from UML to TIPM can compute which UML model element is associated with which model element in an initial MDPE development model (e.g., a Business Process Modelling Notation model). Therefore, the annotations, which are done on, for instance, business process models in the MDPE process, can directly be associated with UML model elements.

This association is computed in an imperative rule in our UML to TIPM transformation code. In order to make that imperative rule work, a trace model is required as an input for our UML to TIPM transformation. This trace model specifies the direct trace links between the development model elements and the UML model used as an input for the transformation. In the case of the UML to TIPM transformation, we simply use the trace model between development models of MDPE and UML generated with the help of a Higher-Order Transformation.

It is obvious that our approach is only useful in the case related target model element(s) are available for the annotated host model element(s) after the execution of an automated model transformation chain. If this is not the case then the domain expert needs to be informed by the tooling we provide since the annotation he did was useless for his purpose.

Use of embedded DSMLs for Performance Assessment Computation in the MDPE process: A second step where annotated information is used is after the *Assessment Computation Actor* (see Figure 2). At this step the generated TIPM as well as the annotated Modification Constraints, Performance Requirements and Performance Objectives are required as input. This input is used to compute the Performance Assessment View, such as the computation, etc., of an optimal configuration, as described in [7]. For instance, we wanted to compute the optimal number of resources for the departments involved in the back-end process used in our case study - Sell-From-Stock-Scenario, which was extended with two additional manual steps to meet the specific needs of a wine seller (see subsection 2.1).

The Modification Constraints, Performance Requirements and Objectives are annotated on the original development models.

However, the Performance Assessment Computation is based on the TIPM, which is simulated using the Anylogic tool as described before. This Performance Assessment Computation needs knowledge about which model elements in the TIPM are associated with the Modification Constraints, Performance Requirements and Objectives.

The megamodel enables us to navigate to the trace model between UML and TIPM and the trace model between our TIPM development models and UML, which are then used in order to generate a new trace model that directly links proprietary and TIPM model elements. This is then utilized by the Performance Assessment Computation in order to relate TIPM model elements with model annotations.

Concluding, in many-step transformation chains, where a number of trace models are required as input, we need the megamodel to be able to navigate from models in a transformation chain to their trace models generated from the Higher-Order Transformations. This is needed to generate a direct trace model from, for instance, the models used as input for transformation step one to models used as output of transformation step three, such as in the MDPE case. Therefore, by making use of megamodel we are able to apply our approach even for longer transformation chains.

6 Related Work

While the usage of DSMLs in MDE is commonly seen as a good thing, there are many unresolved issues when it comes to semantics and the interoperability of DSMLs. Often DSMLs only integrate “somehow” by model transformations. More systematic approaches, like [24, 25], propose more formal semantic integration of DSMLs. As the interest in this area increases, dedicated events emerge just now [26].

On the other hand, model annotations, and in particular UML profiles, are often used for the purpose of defining embedded DSMLs (also called language extensions) [27]. There is also an awareness that these kind of embedded languages have different properties than languages defined from scratch, but there is no common understanding what exactly the (dis)advantages of one or the other are in which scenario.

We are currently not aware of any work that explicitly handles model annotations as DSMLs and regards their specific properties in the context of many-step model transformation chains. We believe that this is an important issue worth investigating to understand beneficial usage scenarios of (embedded) DSMLs in general.

7 Conclusion

We have pointed out the close relationship between the annotation capabilities in classical programming languages with the help of embedded DSLs and model transformation chains in a MDE process with the help of embedded DSMLs. However, we showed that, in particular for automated many-step model transformation chains, specific needs arise with regard to annotations in a MDE process.

Thus, it might happen that information defined at some point is not required in the next immediate transformation step, but in a later one. Also, a direct manipulation of the host models and their metamodel(s) is not an option.

Our approach utilizes embedded DSMLs, trace models and a megamodel. Based on our xWURST case study, concrete examples for each of these modelling artefacts have been given.

We implemented the proposed approach in a tool providing business performance related decision support. We experienced a high value of the approach as we are now able to annotate host models, in particular models conforming to the Business Process Modelling Notation and SAP proprietary models, with, for instance, business performance parameters. This enables us to utilize the Eclipse EMF based “MDPE Workbench” in combination with a SAP proprietary modelling tool and therefore to provide performance related decision support based on real-world business process models.

In future we anticipate gaining more experience with the proposed approach based on other business processes. Furthermore, it is planned to extend the current state of the MDPE process with functionality for guided business performance simulations. An open source version of the "MDPE Workbench" is planned as well.

Disclaimer

The information in this document is proprietary to the following MODELPLEX consortium members: SAP AG and TU-Dresden. 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 TU-Dresden.

References

1. P. Hudak: Building domain-specific embedded languages. ACM Computing Surveys **28**(4es) (1996) 196–196
2. M. Bravenboer, E. Visser: Concrete syntax for objects: domain-specific language embedding and assimilation without restrictions. In: OOPSLA '04: Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, New York, NY, USA, ACM (2004) 365–383
3. OMG: MetaObject Facility (MOF) specification version 2.0 (January 2006) URL http://www.omg.org/cgi-bin/doc?formal/2006-01-01.
4. OMG: Mda guide version 1.0.1 (2003)
5. D.L. Parnas: On the criteria to be used in decomposing systems into modules. In: Communication of the ACM, Vol.15, N12. (1972)
6. M. Fritzsche, J. Johannes: Putting Performance Engineering into Model-Driven Engineering: Model-Driven Performance Engineering. In: MODELS'07 Event LNCS 5002 (Selected Papers), Springer (2007)
7. M. Fritzsche, W. Gilani, I. Spence, T. J. Brown, P. Kilpatrick, R. Bashroush: Towards performance related decision support for model driven engineering of enterprise soa applications. Volume 0., Los Alamitos, CA, USA, IEEE Computer Society (2008) 57–65
8. M. Fritzsche, J. Johannes, S. Zschaler, A. Zherebtsov, A. Terekhov: Application of tracing techniques in model-driven performance engineering. In: 4th ECMDA Traceability Workshop (ECMDA-TW) Proceedings. (2008) 111–120
9. OMG: Business Process Modeling Notation Specification, Final Adopted Specification (2006)
10. SAP AG: Review: SAPPHIRE 2008 - A new star is born in the BPM Galaxy
11. M. Fritzsche, W. Gilani, C. Fritzsche, I. T. A. Spence, P. Kilpatrick, T. J. Brown: Towards utilizing model-driven engineering of composite applications for business performance analysis. In: ECMDA-FA, LNCS 5095, Springer (2008) 369–380

12. M. Altenhofen, T. Hettel, S. Kusterer : Ocl support in an industrial environment. In: Models in Software Engineering. (2007) 169–178
13. XJ Technologies: AnyLogic — multi-paradigm simulation software URL http://www.xjtek.com/anylogic/.
14. A, Knöpfel, B. Gröne, P. Tabeling: Fundamental Modeling Concepts: Effective Communication of IT Systems. John Wiley & Sons (2006)
15. ATLAS Group: AMW Use Case - Model annotations in Java 1.4, URL http://www.eclipse.org/gmt/amw/usecases/annotation (2007)
16. G. Hillairet: AMW Use Case - Metamodel Annotation for Ontology Design, URL http://www.eclipse.org/gmt/amw/usecases/oamusecase (4 2007)
17. B. Vanhooff, S. Van Baelen, W. Joosen, Y. Berbers: Traceability as input for model transformations. In: ECMDA-FA 3th workshop on traceability. (2007)
18. F. Jouault: Loosely Coupled Traceability for ATL. In: ECMDA-FA workshop on traceability. (2005)
19. D. S. Kolovos, R. F. Paige, F. A.C. Polack: On-demand merging of traceability links with models. In: Proceedings: 2. ECMDA-FA workshop on traceability. (2006)
20. Barbero, F. Jouault, J. Bézivin: Model driven management of complex systems: Implementing the macroscope's vision. In: 15th ECBS'08, IEEE (2008) 277–286
21. The AMW Project Team: Atlas Model Weaver (June 2007) URL http://eclipse.org/gmt/amw/.
22. A. Sabetta, D. C. Petriu, V. Grassi,R. Mirandola: Abstraction-raising Transformation for Generating Analysis Models. In: MoDELS'2005 Satellite Events: Revised Selected Papers, LNCS 3844, Springer (2006) 217–226
23. ATLAS Group: ATLAS transformation language (June 2007) URL http://www.eclipse.org/m2m/atl/.
24. M. Bräuer, H. Lochmann: An ontology for software models and its practical implications for semantic web reasoning. In: ESWC. (2008) 34–48
25. K. Chen, J. Sztipanovits, S. Neema: Toward a semantic anchoring infrastructure for domain-specific modeling languages. In: EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software, New York, NY, USA, ACM (2005) 35–43
26. J. Beźivin, A. Pierantonio, L. Tratt, ed. In J. Beźivin, A. Pierantonio, L. Tratt, ed.: Intl. Workshop on Coordination of DSLs, L'Aquila, Italy (September 2008)
27. C. Atkinson, T. Kühne, B. Henderson-Sellers: Systematic stereotype usage. Software and System Modeling **2**(3) (2003) 153–163