A Scalable Approach to Annotate Arbitrary Modelling Languages

Mathias Fritzsche[1], Wasif Gilani[2], Michael Thiele[3], Ivor Spence[4], T. John Brown[5] and Peter Kilpatrick[6]

Abstract: Refinement via annotations is a common practice in Model-Driven Engineering (MDE). For instance, in the case of our Model-Driven Performance Engineering (MDPE) architecture, we are required to annotate different types of process models with performance objectives, constraints and other information. This is used to enable domain experts, such as business analysts, to benefit from an automated performance prediction based decision support.

Currently, the process models are annotated manually, element by element. This approach is not scalable, for instance, in the case where numerous model elements in large model repositories need to be annotated with the same information. Thus, a scalable annotation mechanism is needed which can be used for arbitrary modelling languages. In this paper we propose an architecture which uses a specialized modelling language to express annotations in an efficient way. This language is transformed to model transformation scripts in order to generate annotation models, which separate the annotated information from the target models and, therefore, supports scalable model annotations for modelling languages of choice.

1 Introduction

Refinement of models is a vital component of Model Driven Engineering (MDE) in order to decrease the level of abstraction vertically, for instance, by refining a business process model as an Enterprise Java Bean (EJB) based application. Such vertical refinements enable separation of platform-specific information, such as the middleware used, from platform-independent concerns. Moreover, horizontal refinement is required in order to move from one view point of the system to another. In our previous work [FPG+09] we presented an example of horizontal refinement. That example showed a stepwise transformation chain from business process models representing the business behaviour, to simulation models which combine business behaviour with performance related behaviour.

For all kinds of refinements, in the first step, an annotation of additional information to the target model, e.g., a Business Process Modelling Notation (BPMN) model [Obj06],

[1] SAP Research CEC Belfast, mathias.fritzsche@sap.com
[2] SAP Research CEC Belfast, wasif.gilani@sap.com
[3] Technische Universität Dresden, michael.thiele@inf.tu-dresden.de
[4] Queen's University Belfast, i.spence@qub.ac.uk
[5] Queen's University Belfast, tj.Brown@qub.ac.uk
[6] Queen's University Belfast, p.kilpatrick@qub.ac.uk

is required. In the second step, the annotated (enriched) target model is used, for instance, as input for a code generator or other transformation steps. The annotation can conform to a profile definition, such as proposed for the Unified Modelling Language (UML). Profile definitions are extensions of the target meta-model as described in the UML standard [Obj07]. However, some authors have presented annotation technologies [VCFM, VGK07, FJA+09] where the meta-models of annotations are defined without extending the target meta-model.

We experienced the problem that a systematic and generic approach to the manually performed annotation task itself has hardly been considered yet if annotations are cross-cutting. Instead, target model elements are normally annotated manually, element by element. If numerous model elements in large model repositories need to be annotated with the same information, this becomes a vast manageability problem. Moreover, a solution needs to support arbitrary target modelling languages.

In this paper we propose the combined use of a generic meta-model for model annotations, which can be applied to annotate modelling languages of choice, and a Domain Specific Language (DSL) describing the model annotations at a high level of abstraction, i.e. the user is unaware of the actual underlying employed annotation technology. This annotation script is translated to the Atlas Transformation Language (ATL) [JABK08] in order to use the ATL execution engine to interpret the script and to generate annotation models. This approach complements existing manual annotation editors for cases of cross-cutting annotations.

This paper is structured as follows: Section 2 demonstrates the need for model annotations based on an industrial case study called Model-Driven Performance Engineering (MDPE). In Section 3 we motivate the need for an efficient and scalable way of carrying out model annotation. Section 4 gives an overview of a generic annotation meta-model that we utilized for the solution which is provided in Sections 5. In Section 6 we describe the experience gained with our approach in the context of MDPE. A comparison with the current state of the art is provided in Section 7. Section 8 concludes the paper.

2 Use Case: Model-Driven Performance Engineering

It is the goal of Model-Driven Performance Engineering (MDPE) [FPG+09,FJ07,FGF+08, FBV+09] to provide performance related decision support to domain experts (e.g. business analysts) based on the models they understand, such as BPMN models [Obj06]. The decision support provided by MDPE is currently based on a simulation engine and an optimization engine. It enables answers to questions such as:

- Can available staff cope with the future business growth? This can be simulated by increasing the planned number of business process instances that are intended to be executed in a given time.

- Will training of employees be sufficient to achieve certain business performance targets? This can be simulated by decreasing the net working time for certain process steps.
- Is the hiring of additional employees inevitable? This can be simulated by increasing the resource capacities.
- How many employees are needed at which point of time? This can be answered by an optimization engine which automatically simulates a number of combinations and selects the best combination based on given requirements, objectives and constraints [FGS+08].

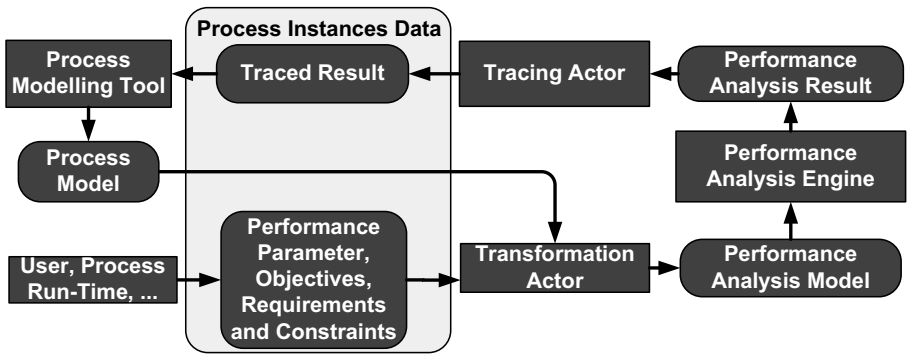

Fig. 1: Model Driven Performance Engineering as a Block Diagram [KGT06]

As shown in Figure 1, MDPE enables automatic generation of *Performance Analysis Models* from different types of *Process Models* conforming to different meta-models. Thus, MDPE extends existing *Process Modelling Tools*, such as the BPMN [Obj06] based tool NetWeaver BPM [SRMS08], other SAP proprietary process modelling tools and JPASS [jCO09]. Different model repositories are employed for storing the *Process Models* and their meta-models. Therefore, in the current MDPE implementation the Eclipse Modelling Framework (EMF) [BBM03] and the SAP Modelling Infrastructure (MOIN) [AHK06] are supported. The meta-models in the EMF repository conform to the ECORE meta-modelling language whereas meta-models in the MOIN conform to the MOF 1.4 [Obj02] meta-modelling language.

Generated *Performance Analysis Models* are used as input for the *Performance Analysis Engine*, such as the AnyLogic simulation engine [XJ 09]. *Performance Analysis Results* are traced back and visualized based on the original *Process Models* the domain expert understands, as described in [FJZ+08, FJA+09].

The generation of the *Performance Analysis Models* is done via model to model transformations. It is shown by Figure 1 that these transformations also require *Performance Parameters*, which are values specifying the resource related behaviour of the modelled process instances (see *Process Instance Data* in Figure 1) over a period of time. Examples are planned or historic workload of a process (e.g., number of arriving Sales Orders in a

Sales Order Processing business process). In order to use not only simulation engines as Performance Analysis Engine, but also other ones such as the optimization engine OptQuest [Rog02], MDPE additionally takes *Objectives, Requirements and Constraints* into account as proposed in [FGS+08].

All this data, together with the *Performance Parameter* and *Traced Results*, are added as annotations to the *Process Models* as discussed in the following section.

3 Identified Problem

At this point of the paper, MDPE has been introduced as an example where model annotations are heavily used.

Current editors for model annotations, such as the editors used for UML profiles, require users to manually select each single model element to enrich it with additional data, such as planned performance parameters, objectives, requirements or constraints in the case of MDPE. This is a time consuming and error prone task.

This manual annotation becomes a significant problem in large models [GLZ06, MF07] in cases where multiple model elements have to be annotated with the same cross-cutting information. For instance, the same business performance requirements in MDPE are often valid for a number of business process steps within a process model. An example is the requirement that all approval steps in a complex business process need to be processed within 2 days. All business process model elements named “Approval” need to be annotated with this requirement.

Concluding mechanism is needed that complements existing manual annotation approaches to carry out cross-cutting model annotation in an efficient, automated and scalable way. This mechanism needs to support modelling languages of choice, such as BPMN, UML, JPASS, etc.

In the following two sections a solution for this problem is provided. First, our approach for non-intrusive annotation of arbitrary modelling languages is summarized. Second, in Section 5, our approach to express annotations in a scalable manner, and to automatically generate annotation models is presented.

As an example we use a BPMN model of a sales business process which is executed by a specific sales unit. We perform the annotation in order to simulate a scenario in which all “Approval” steps in this business process have to be processed by the “ManagerYang”.

4 An Annotation Approach for the Modelling Language of Choice

Like others [D’A05, GP02] we initially used UML profiles [Obj07] for the annotation of additional data to UML Activity Diagrams in our initial version of the MDPE architecture. This enabled us to gain experience with the automatically generated performance models

[FGF+08]. The benefit of this approach was that the tool support, which is available for most UML editors generically, could be utilized for our MDPE specific profiles.

We, however, had to deal also with modelling languages other than UML, such as BPMN [Obj06] conformant models like JPASS [Fle95, jCO09] models and SAP proprietary models. Additionally, because profiles are a meta-model extension mechanism, one needs "write access" to the related meta-models, which is sometimes not possible. This particularly became obvious when we applied the MPDE approach to SAP proprietary models where we were not able to modify the meta-models, as described in [FJA+09]. Thus, the application of a profile-like concept for annotation in the case of SAP proprietary models is not possible. Therefore, in [FJA+09] we defined an alternative solution to UML profiles which can be applied to all modelling languages of choice.

Our approach for model annotations is based on so-called annotation models, as described in [VGK07]. Annotation models enable the definition of separate models containing references to target models. Since the annotation information is encapsulated in its own space, this approach enables the enrichment of arbitrary (process) modelling languages with additional information without polluting them. Moreover, in cases where multiple target model elements are annotated with the same information, only one annotation model element needs to be created.

In order to provide generic tool support, e.g., for editing annotations, we were required to define a basic structure which is refined by concrete annotation meta-models. Therefore, we defined a generic annotation meta-model called *Annotation Meta-Model (AMM)* [FJA+09], as seen in the middle part of Figure 2. The AMM itself inherits from the weaving meta-model *MWCore* provided by the ATLAS group [FBV06] (see upper part of Figure 2). This weaving meta-model enables the definition of links between models independent of any specific modelling language, such as UML.

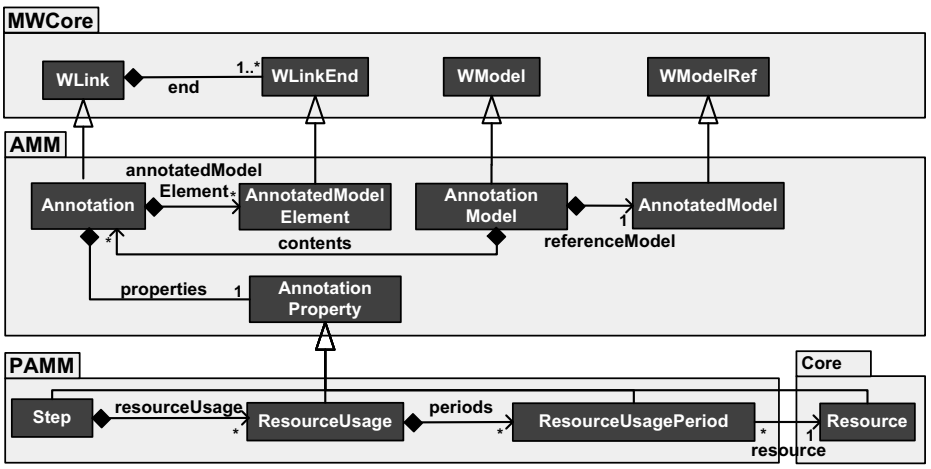

Fig. 2: Generic Annotation Meta-Model (AMM, middle part), and extract of a specific annotation meta-model for performance parameters (Parameter Annotation Meta-Model (PAMM), lower part)

The AMM refines the meta-class *WLink* from the *MWCore* meta-model with the meta-class Annotation. The *WLink* class is indirectly associated with an attribute *"ref"* of type String, which is used to represent references in the enriched models, such as Process Models used in MDPE. Hence, the annotation model elements represent the weaving link to the target model elements. Additionally, an annotation contains multiple *AnnotationProperties*, which represent the annotated information to the source model elements.

An *AnnotationProperty* can be further refined for the needs of a specific annotation meta-model for a certain domain, as shown by the lower part of Figure 2. In the visualized example we refined the AMM for the specific needs of MDPE. A *Step* references a *ResourceUsage* which further references a *ResourceUsagePeriod*. Such a period is linked with a specific *Resource*, such as "ManagerYang". This simple example enables annotation of resource usages to steps over a period of time in a Process Model.

In order to achieve reusability in our annotation meta-model, it is possible to distribute *AnnotationProperties* among packages, such as the "PAMM" (Parameter Annotation Meta-Model) for performance parameters and the "Core" package as shown in Figure 2. In our example, the "Core" package encapsulates the meta-classes, which are referenced within other packages.

To provide generic tool support based on the AMM within the Eclipse environment, the extension of the Annotation Meta-Model can be done by implementing an Eclipse Extension-Point. Thus, a user of the generic AMM based annotation approach can apply annotations to selected model elements in a similar way to the use of UML profiles within model editors like Topcased [The09].

However, with the generic graphical annotation editor it is still necessary to manually select each target model element in order annotate it with additional information.Thus, even if the AMM separates model annotations from the target models and therefore enables annotation of arbitrary modelling languages, a mechanism is still needed to introduce scalability and automation in order to reduce the time needed by the user to define numerous model annotations. This mechanism therefore need to complement the existing annotation editor for cases of cross-cutting annotations.

5 Scalable Model Annotations based on the AMM

For the automated annotation of multiple model elements we define annotations within a separate script. Figure 3 shows that this script includes a *query* phase that defines which model elements should be annotated, and an *execution* phase that defines and creates the annotations for the queried elements (see "Script"). Triggered through user input, both phases are executed (see "Query and Annotation Execution"). As a result, the new annotations with references to the target model are created, such as AMM based annotations (see "Annotation Model").

In the following subsections we outline an approach to implement of the proposed solution.

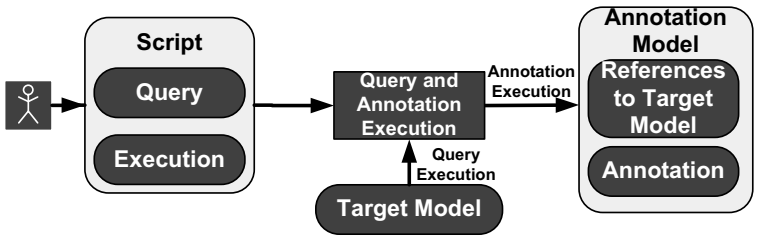

Fig. 3: Proposed solution for non-intrusive model annotation as Block Diagram [KGT06]

5.1 Use of General Purpose Model Transformation Languages

Gray et al. [GLZ06] also proposes to address scalable change evolution for MDE via scripts containing information for refining existing models. He suggests general purpose transformation languages to be used as scripting languages for such refinements. Utilization of such languages for scalable model annotations would result in the following benefits:

- Some general purpose transformation languages abstract from the underlying model repository as long as there is an adapter for the repository. This is required for the MDPE case (see Section 2).
- To implement the *query* phase, transformation languages support an element selection mechanism (most languages offer OCL-based guards or selections on collections of possible elements). Since we support multiple repositories, we cannot expect that all of them offer such a high-level query mechanism.

Summarising, the functionality offered by model transformation languages is sufficient for an annotation language. However, compared to Gray's work, we only wish to create annotation models that conform to meta-models which are based on the generic AMM meta-model.

When we tried to write transformation scripts to create AMM based annotations for arbitray modelling languages, we experienced severe problems with the transformation script based approach. This was due to the fact that the user has to know the generic meta-model of the AMM and create all of its elements such as the *Annotation* elements by hand until the actual *AnnotationProperty* can be created. This is a repetitive and therefore tedious and error-prone task.

Moreover, the syntax of transformation languages is tailored to the general task of model transformations and not for the rather specific task of model annotations. This different purpose again results in intricate code. For instance, ATL's syntax is much broader than what the annotation problem requires and therefore complicates the task of finding a solution.

Concluding, a purely transformation script based approach is not efficient. For instance, in order to specify that all *ManualActivities* with the name "Approval" and which are executed in a specific sales unit have to be annotated with a specific resource consumption from the resource "ManagerYang", a number of *AnnotationProperties* have to be created, namely a number of *Steps* which have references to the *Resource* "ManagerYang". Implementing this script in the Atlas Transformation Language (ATL) requires nearly 150 lines of ATL code and we had to make use of highly ATL-specific functions. A more detailed description of the experiences that we gained with ATL is provided in the next section.

However, a specialized DSL for AMM based model annotations on top of the general purpose transformation language could be employed based on certain assumptions about the AMM based structure of the annotation. Therefore, the effort required by a user for model annotations can be significantly reduced, by still providing an AMM based model annotation approach which permits usage for arbitray modelling languages. In the following section such a DSL is provided.

5.2 A DSL based approach: EQUAL

The benefits of DSLs are manifold. Consel [CM98] and Czarnecki [CVS+06] mention that a DSL is needed in those cases where an isolated problem, such as the automated model annotation, needs to be addressed which could occur again in the future. Furthermore, Jouault [JBK06] claims that with DSLs domain concepts can be represented by constructs of the language and that these constructs usually are high level. Thus, if a construct of a DSL is mapped to a construct of a General Purpose Language (GPL) the DSL construct is typically significantly shorter.

Thus, a goal of a DSL should be to express the task in a concise way and it should hide as much detail as possible from the user of the DSL. Based on this design principle we hide the complex AMM construction and the explicit creation of *Annotation*. That led to a concise declarative way to describe model annotations, which is not possible with transformation languages, since - even if the transformation language itself is declarative - multiple instructions are required to create the annotation. Hence, the syntax can be more tailored and contains fewer elements than a model transformation language offers. We call this AMM based language *EQUAL* - Extended QUery and Annotation Language.

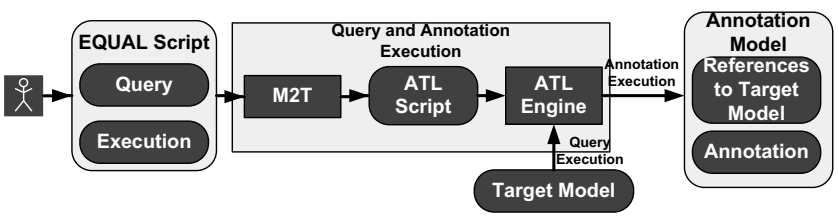

Fig. 4: ATL based architecture for non-intrusive model annotation solution as Block Diagram [KGT06]

The overall architecture of our approach is depicted in Figure 4. The user writes an EQUAL script in a textual editor. This script contains target model queries and *AnnotationProperties*. It is then transformed into ATL scripts that are interpreted. The ATL scripts query the target model and create or alter annotations in the affected annotation model elements.

The EQUAL interpreter is configured with a separate *Configuration* script which is shown in the following listing. This part corresponds to the configuration part of an ATL transformation, i.e. one has to first tell the target meta-models and models. Additionally, it enables to tell the meta-elements which are used for the model annotation. This enables better tool support for the EQUAL editor that we have implemented, especially in terms of auto completion.

```
1  metamodels  {
2    BPMN_NetWeaver : MOIN from 'local:\\';
3    PAMM : EMF from 'uri:http://www.modelplex/amf/2009/MDBPE/PAMM';
4    CORE : EMF from 'uri:http://www.modelplex/amf/2009/MDBPE/CORE';
5  }
6  models  {
7    salesProcessOpportunityManagement : BPMN from 'local:\\
         SalesProcessOpportunityManagement';
8    pam : PARAMETER : from 'ext:C:\\Ann2\\Parameter.mdpe';
9    core : CORE from : from 'ext:C:\\Ann2\\Core.mdpe';
10 }
11 types  {
12   ManualActivity : BPMN_NetWeaver;
13   Step : PAMM;
14   ResourceUsage : PAMM;
15   ResourceUsagePeriod : PAMM;
16   Resource : CORE;
17 }
```

The following listing shows an example of an EQUAL script. The concrete syntax of this script is provided in the appendix. In the following example it can be seen that it consist of a *query* part (lines 1-3 in the listing) and an *annotation* part (lines 4-12 in the listing).

```
1  query  {
2    manualActivities = SELECT ManualActivity FROM
         salesProcessOpportunityManagement WHERE name = "Approval";
3  }
4  annotate  {
5    manualActivities with Step.new  (
6    resourceUsage.add (
7      resource = SELECT Resource FROM core WHERE name = "ManagerYang",
8      periods.add (
9        ResourceUsagePeriod.new (
10       startDate = "1.1.2010",
11       netWorkingTimeConsumption = 0.15))));
12 }
```

The query part corresponds to the point cut expressions in aspect oriented languages [KHH+01, SGSP02]. It can be seen in the listing that we choose a SQL like syntax for these queries. The resulting set of model elements of a query defines the target model ele-

ments that have to be annotated in the annotation part of the script. The example shows the query to get all these *ManualActivitie*s in a BPMN model which are called “Approval”.

The annotate part of the script is similar to the aspect code in AOP [KHH+01, SGSP02]. In the example provided by the listing, attributes of the new annotation are set. All manual activities with the name “Approval” are annotated with a *resourceUsage* of the “ManagerYang” by referencing an existing model element. This reference is enabled, again, via the support of SQL-like queries.

The EQUAL script and the Configuration script of the previous listings are transformed into ATL scripts via a model-to-text transformation. In a first transformation step, the *query* part of the EQUAL script is transformed into an ATL transformation script that queries the target model elements and additionally determines which of these elements were not annotated before. We need to store the *ref* String for those target model elements, since new annotations, created for these elements, should reference them (see Section 4, *ref* attribute). Unfortunately, we had to implement this process in a separate transformation step for reasons described in Section 6.2.

After the execution of this ATL script it is known how many new annotations have to be created and which target model elements they reference. This information is used to generate a subsequent ATL transformation that performs the actual annotation (*execution* phase), since here, new annotations have to be created for every queried model element that was not annotated before.

6 Experiences Gained

Based on our implementation of EQUAL DSL, and the underlying architecture, including the evaluation of the approach for MDPE, we have gained a number of experiences.

6.1 Implementation and usage of the EQUAL DSL

The “openArchitectureWare” framework [ope09] allowed us to easily generate an Eclipse based editor for our DSL that offers user assistance such as syntax highlighting and code completion out of the box or with little customization. It also facilitates checking of conditions. Thus, input errors can easily be detected in the EQUAL editor. This increases further the usability of the EQUAL editor. This is a clear advantage over a pure general purpose transformation language based approach where editors, such as the ATL editor, are not able to provide hints for special use cases, such as MDPE, as they simply allow a wider range of expressions than EQUAL.

Additionally, we compared the efficiency of using EQUAL for model annotations with carrying out the model annotations manually with an annotation editor. Compared to the manual annotation, we particularly experienced benefits using EQUAL when we had to annotate numerous model elements of large target models with the same cross cutting

information, such as the annotation of 26 "Approval" steps in a model containing 772 model elements. In such a case, writing a textual EQUAL script is less repetitive and therefore more timesaving than doing the annotation manually.

6.2 Use of ATL as an Execution Language

We experienced that ATL, which is only one of many possible model transformation languages, is well-suited as an execution language for scalable model annotations due to its powerful language concepts. One example is the extension of the OCL-defined operations for strings, as this includes the use of regular expressions. Hence, it was possible to map regular expressions that appear in an EQUAL query to ATL's OCL expressions.

However, we recognized that the language does not perfectly fit our needs: Due to the possible deep structuring of AMM based annotation meta-models we are forced to create multiple elements and their relations to one another. Therefore, we had to utilize many ATLs rather advanced language features such as the *resolveTemp* method to reference created elements of other rules.

Additionally, refinement transformations have proven to be very useful in our case, since we only need to create new annotations or extend existing ones in an annotation model and do not wish to perform a "real" model transformation with different source and target models. The drawback of this design choice is that the refinement transformations implemented in ATL do not permit the use of imperative code. If we had been able to use such a feature in ATL it would have been possible to iterate over the collection of queried target model elements.

For elements that were not annotated before, we would create new annotations through am imperatively called rule. Since this is not possible with the refinements transformation support of the ATL version used, we had to implement distinct ATL scripts for the *query* phases as well as for the *execution* phase. The first ATL script has to query the target model and to find out which elements need new annotations. This information is needed as an input for the generation of the second ATL script, as we explicitly have to create ATL code for each new annotation. Concluding, the *query* and *execution* phase have to be executed one after another, i.e. in an imperative manner. This concept cannot be mapped to a declarative language which forces construction of an "imperative bridge" between two declarative phases.

This leads to a crucial side effect considering the amount of ATL code to be written for annotations. While the EQUAL script in the given example in the previous section consists of 12 lines of code to create 6 annotations, the generated ATL files contain more than 150 lines of code.

This is verified by another example that we experienced based on MDPE. We ran a query on a model containing 772 elements of which 26 were annotated with the same annotation as in the previous example. The EQUAL script again consisted of 12 lines of code, whereas

the ATL scripts - due to the high number of annotations to be created - contain more than 900 lines of code.

This discrepancy is due mainly to the fact that we have to explicitly create all annotations for each queried target model element and this grows with the number of queried elements. Thus, with the current implementation, ATL in refinement mode cannot be considered to be a scaling model annotation language, but since the ATL code is generated this limitation is not relevant to us. If we would not need to create ATL code for each annotation, we would need less ATL code.

However, even with the possibility of using imperative code in refinement transformations, for instance by using other languages than ATL, the transformation script still would be longer and significantly more complex than the corresponding EQUAL script, as the AMM elements would be visible and the annotation process could not be described in a fully declarative manner.

6.3 DSL Stacking

Although ATL is a DSL, its domain (model transformations) is not specific enough for our task at hand. We showed that making it more domain-specific (model annotations) through the use of another DSL on top of it greatly helps to reduce the coding effort to solve our domain-specific problem. This "DSL stacking" is comparable to the notion of Platform Independent Model (PIM) and Platform Specific Model (PSM) in the MDA architecture of the OMG [Obj03].

7 Related Work

Related to our approach there are a number of aspect oriented languages such as AspectJ [KHH+01], AspectC++ [SGSP02], etc. which offer special constructs and mechanisms like pointcuts and joinpoints to express and weave cross-cutting code back into a target language.

A number of approaches for model annotations, such as UML profiles [PSS07, XWP03, BM05] or model weaving based approaches [VCFM, VGK07] are available. However, scalable model annotations have hardly been considered yet. There is, however, work available to address scalable change evolution for MDE [GLZ06] via general purpose transformation languages.

Compared to that, we propose defining model annotations in their own space with a specialized DSL. This DSL takes advantage of a given annotation structure provided by the generic AMM which enables reducing the number of expressions to be defined by the user. Our approach would also be applicable for other generic annotation approaches, such as provided by UML profiles, but the AMM enables annotating any kind of model conforming to any kind of meta-modelling language. For example, this is needed to apply MDPE

for complex business processes which are orchestrated out of back-end processes, with a long life-cycle and composite processes, with a short life-cycle, where both kinds of processes conform to different meta-models.

Using domain specific (modelling) languages instead of GPLs is not a new idea. A number of examples of applying them for a number of different domain specific problems can be found in the literature since the 1990s [ADR95, GGJ+97, KH97]. Nowadays, graph-based meta-modelling languages, such as MOF [Obj02] are available. Additionally, sufficient tool support for the efficient implementation and management of DSL tools, such as the tools contained in the AMMA open source Eclipse platform [KBJV06], can be utilized.

However, to our knowledge only the approach proposed by Mehr [MF07] addresses the problem of providing a specialized language for systematic model annotations of UML models. Compared to that, our approach can be applied for any target modelling language. Furthermore, we are transforming our DSL to a general purpose transformation language in order to utilize its execution engine. The same concept is applied by state of the art model to code transformations. This enables the utilization of existing code execution engines, such as the Java Virtual Machine (JVM), for modelling languages.

8 Conclusion

Based on the numerous process model annotations required for MDE specialized processes, such as the MDPE process, the need for a systematic model annotation approach is presented. We have presented the combined use of the EQUAL DSL and automatically generated ATL model transformation scripts to automate model annotations which enables utilization of the ATL execution engine. However, we showed that using our DSL for model annotations is much more efficient than manually writing ATL scripts.

Moreover, the generated annotations are represented by a separate AMM conforming model which separates model annotations from the target models and therefore permits the application of the EQUAL approach for arbitrary modelling languages.

We showed that, if an annotations is cross-cutting, EQUAL enables users to perform model annotations in a much more efficient way than doing it manually via state of the art model annotation editors. Thus, modellers are now able to employ our approach as an additional annotation tool when it is necessary to automatically annotate numerous model elements with the same data. Elements can be queried based on their type, name or previously applied annotations. The last query type furthermore enables us to trigger annotations based on already existing annotations which enables stepwise refinement.

As future work we anticipate extending our approach to annotate model elements based on a wider range of algorithms, for instance, to decrease all annotated processing time targets of a business process model by 10%.

References

[ADR95] B. R. T. Arnold, A. Van Deursen, and M. Res. An algebraic specification of a language for describing financial products. In *Proceedings of the IEEE Computer Society Workshop on Formal Methods Application in Software Engineering*, pages 6–13, 1995.

[AHK06] Michael Altenhofen, Thomas Hettel, and Stefan Kusterer. OCL Support in an Industrial Environment. In *Models in Software Engineering: Workshops and Symposia at MODELS 2006, Reports and Revised Selected Papers*, volume 4364 of *LNCS*, pages 169–178, 2006.

[BBM03] Frank Budinsky, Stephen A. Brodsky, and Ed Merks. *Eclipse Modeling Framework*. Pearson Education, 2003.

[BM05] Simonetta Balsamo and Moreno Marzolla. Performance evaluation of UML software architectures with multiclass Queueing Network models. In *Proceedings of the 5th International Workshop on Software and Performance (WOSP'05)*, pages 37–42. ACM, 2005.

[CM98] Charles Consel and Renaud Marlet. Architecturing software using a methodology for language development. In *Proceedings of the 10th International Symposium on Programming Language Implementation and Logic Programming*, volume 1490 of *LNCS*, pages 170–194, 1998.

[CVS+06] Krzysztof Czarnecki, Markus Völter, Thomas Stahl, Jorn Bettin, Arno Haase, Simon Helsen, and Bettina von Stockfleth. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[D'A05] Andrea D'Ambrogio. A model transformation framework for the automated building of performance models from UML models. In *Proceedings of the 5th International Workshop on Software and Performance (WOSP'05)*, pages 75–86. ACM, 2005.

[FBV06] Marcos Didonet Del Fabro, Jean Bézivin, and Patrick Valduriez. Weaving Models with the Eclipse AMW plugin. In *Proceedings of the Eclipse Modeling Symposium, Eclipse Summit Europe*, 2006.

[FBV+09] Mathias Fritzsche, Hugo Bruneliere, Bert Vanhoof, Yolande Berbers, Frédéric Jouault, and Wasif Gilani. Applying Megamodelling to Model Driven Performance Engineering. In *Proceedings of the International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2009)*, pages 244–253. IEEE Computer Society, 2009.

[FGF+08] Mathias Fritzsche, Wasif Gilani, Christoph Fritzsche, Ivor Spence, Peter Kilpatrick, and Thomas. J. Brown. Towards Utilizing Model-Driven Engineering of Composite Applications for Business Performance Analysis. In *Proceedings of the 4th European conference on Model Driven Architecture Foundations and Applications (ECMDA-FA'08)*, volume 5095 of *LNCS*, pages 369–380. Springer-Verlag, 2008.

[FGS+08] Mathias Fritzsche, Wasif Gilani, Ivor Spence, T. John Brown, Peter Kilpatrick, and Rabih Bashroush. Towards Performance Related Decision Support for Model Driven Engineering of Enterprise SOA Applications. In *Proceedings of the 15th Annual IEEE Computer Society International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'08)*, pages 57–65. IEEE Computer Society, 2008.

[FJ07] Mathias Fritzsche and Jendrik Johannes. Putting Performance Engineering into Model-Driven Engineering: Model-Driven Performance Engineering. In *Models in Software Engineering: Workshops and Symposia at MODELS 2007, Reports and Revised Selected Papers*, volume 5002 of *LNCS*. Springer, 2007.

[FJA+09] Mathias Fritzsche, Jendrik Johannes, Uwe Assmann, Simon Mitschke, Wasif Gilani, Ivor Spence, JohnBrown, and Peter Kilpatrick. Systematic Usage of Embedded Modelling Languages in Automated Model Transformation Chains. In *Proceedings of the 1st International Conference on Software Language Engineering (SLE'08), Revised Selected Papers*, volume 5452 of *LNCS*, pages 134–150. Springer-Verlag, 2009.

[FJZ+08] Mathias Fritzsche, Jendrik Johannes, Steffen Zschaler, Analtoly Zherebtsov, and Alexander Terekhov. Application of Tracing Techniques in Model-Driven Performance Engineering. In *Proceedings of the 4th ECMDA Traceability Workshop (ECMDA-TW)*, pages 111–120, 2008.

[Fle95] Albert Fleischmann. Distributed Systems, Software Design & Implementation. Springer-Verlag, 1995.

[FPG+09] Mathias Fritzsche, Michael Picht, Wasif Gilani, Ivor Spence, John Brown, and Peter Kilpatrick. Extending BPM Environments of your choice with Performance related Decision Support. (Best Paper Award). In *Proceedings of the 7th Business Process Management Conference (BPM'09)*, volume 5701 of *LNCS*, pages 97–112. Springer-Verlag, 2009.

[GGJ+97] Neeraj K. Gupta, Neeraj K. Gupta, Lalita Jategaonkar Jagadeesan, Lalita Jategaonkar Jagadeesan, Eleftherios E. Koutsofios, Eleftherios E. Koutsofios, David M. Weiss, and David M. Weiss. Auditdraw: Generating audits the FAST way. In *In Proceedings of the 3rd IEEE Computer Society International Symposium on Requirements Engineering*, pages 188–197, 1997.

[GLZ06] Jeff Gray, Yuehua Lin, and Jing Zhang. Automating Change Evolution in Model-Driven Engineering. In *Computer*, volume 39, pages 51–58. IEEE Computer Society, 2006.

[GP02] Gordon P. Gu and Dorina C. Petriu. XSLT transformation from UML models to LQN performance models. In *Proceedings of the 2th International Workshop on Software and Performance (WOSP'02)*, pages 227–234. ACM, 2002.

[JABK08] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. ATL: A model transformation tool. In *Science of Computer Programming*, volume 72, pages 31–39, 2008.

[JBK06] Frédéric Jouault, Jean Bézivin, and Ivan Kurtev. TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering. In *Proceedings of the 5th international conference on Generative programming and component engineering (GPCE'06)*, pages 249–254. ACM, 2006.

[jCO09] jCOM1 AG. jPASS! - Subjektorientierte Prozessmodellierung. `http://www.jcom1.com/cms/jpass.html`, 2009.

[KBJV06] Ivan Kurtev, Jean Bézivin, Frédéric Jouault, and Patrick Valduriez. Model-based DSL frameworks. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications (OOPSLA'06)*, pages 602–616. ACM, 2006.

[KGT06] Andreas Knöpfel, Bernhard Gröne, and Peter Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems.* John Wiley & Sons, 2006.

[KH97] Samuel N. Kamin and David Hyatt. A Special-Purpose Language for Picture-Drawing. In *Proeedings of theConference on Domain-Specific Languages*, pages 297–310. Usenix, 1997.

[KHH+01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of AspectJ. In *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP'01)*, volume 2072 of *LNCS*, pages 327–353. Springer-Verlag, 2001.

[MF07] Farid Mehr and Mathias Fritzsche. Patent application: Annotation of models for model-driven engineering, Attorney Docket No. 2007P00439US/0010-076001, 2007.

[Obj02] Object Management Group. MetaObject Facility (MOF) Specification Version 1.4. `http://www.omg.org/technology/documents/formal/mof.htm`, 2002.

[Obj03] Object Management Group. MDA Guide Version 1.0.1. `www.omg.org/docs/omg/03-06-01.pdf`, 2003.

[Obj06] Object Management Group. Business Process Modeling Notation Specification, Final Adopted Specification, Version 1.0. `http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf`, 2006.

[Obj07] Object Management Group. Unified Modeling Language (OMG UML), Infrastructure, V2.1.2. `http://www.omg.org/spec/UML/2.1.2/`, 2007.

[ope09] openArchitectureWare. `http://www.openarchitectureware.org/`, 2009.

[PSS07] Dorina Petriu, Hui Shen, and Antonino Sabetta. Performance analysis of aspect-oriented UML models. In *Software and Systems Modeling*, volume 6, pages 453–471, 2007.

[Rog02] Paul Rogers. Optimum-seeking Simulation in the Design and Control of Manufacturing Systems: Experiences with OptQuest for Arena. In *Proceedings of the 2002 Winter Simulation Conference (WSC'02)*. Winter Simulation Conference, 2002.

[SGSP02] Olaf Spinczyk, Andreas Gal, and Wolfgang Schröder-Preikschat. AspectC++: An Aspect-Oriented Extension to C++. In *Proceedings of the 40th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific)*, February 2002.

[SRMS08] Jim Hagemann Snabe, Ann Rosenber, Charles Mølle, and Mark Scavillo. *Business Process Management: The SAP Roadmap.* SAP Press, 2008.

[The09] The Topcased Project Team. Topcased. `http://www.topcased.org`, 2009.

[VCFM] Juan M. Vara1, Valeria De Castro, Marcos Didonet Del Fabro, and Esperanza Marcos. Using Weaving Models to automate Model-Driven Web Engineering proposals. In *Proceedings of Integración de Aplicaciones Web (ZOCO'08)*, pages 86–95.

[VGK07] Markus Völter, Iris Groher, and Bernd Kolb. Mechanisms for Expressing Variability in Models and MDD Tool Chains. In *Proceedings of the SIG-MDSE-Workshop on MDSD in Embedded Systems*, 2007.

[XJ 09] XJ Technologies. AnyLogic — multi-paradigm simulation software. `http://www.xjtek.com/anylogic/`, 2009.

[XWP03] Jing Xu, Murray Woodside, and Dorina Petriu. Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time. In *Proceedings of the 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS 03)*, volume 2794 of *LNCS*, pages 291–310. Springer-Verlag, 2003.

Appendix: EQUAL Syntax

In this appendix, the Xtext syntax [ope09] of the EQUAL language is presented.

```
QUAL :
  queryPart=QueryPart
  application=AnnotationPart;

QueryPart :
  "query" "{" name=ID "=" query=Query "}";

Query :
  "SELECT" type=[Type|TypeName] ("and" type=[Type|TypeName] )*
  "FROM" model=[Model] ("and" model=[Model] )*
  ("WHERE" (where=OrExpression))? ";";

String TypeName :
  ID | STRING;

OrExpression :
  andExpressions+=AndExpression
  ("OR" andExpressions+=AndExpression)*;

AndExpression :
  notExpressions+=NotExpression
  ("AND" notExpressions+=NotExpression)*;

NotExpression :
  (not?="NOT")?
  (comparatorExpression=ComparatorExpression |
  parExpression=ParExpression);

ParExpression :
  "(" orExpression=OrExpression ")";

ComparatorExpression :
  left=Feature (comparator="=" | comparator="LIKE" | comparator="<" |
  comparator=">" | comparator="<=" | comparator=">=") right=Value;

Feature :
  name=ID (isMultiple?="[" (index=INT | helper=[Helper] |
  isAll?="all") "]")? ("." nextFeature=Feature)?;

Value :
  IntValue | DoubleValue | StringValue | EnumValue |
  HelperValue | RefValue;

IntValue :
  value=INT;

DoubleValue :
  value=DOUBLE;

StringValue :
  value=STRING;

EnumValue :
```

```
54    value=ENUM;
55
56  HelperValue :
57    value=[Helper];
58
59  RefValue :
60    "ref" value=[ComplexFeatureInstance|RefId];
61
62  Native DOUBLE :
63    "RULE_INT'.'('0'..'9')+";
64
65  Native ENUM :
66    "'#'RULE_ID";
67
68  AnnotationPart :
69    "{" ("annotate" toAnnotate=[Query]
70        "with" annotationProperty=[Type] (
71            isNew?= ".new" | (".adaptAll"
72            ("WHERE" where=OrExpression)?) )
73        "(" ((featureInstances+=FeatureInstance)
74            ("," featureInstances+=FeatureInstance)*)? ")" ";" )* "}";
75
76  FeatureInstance :
77    name = Query | ID (
78        isMultiple?="[" (index=INT | isAll?="all")? "]" ) ?
79        ( isAdd?= ".add"  "("
80            featureInstanceValues+=FeatureInstanceValue
81            ("," featureInstanceValues+=FeatureInstanceValue)* ")"  |
82            "=" value=Value | ":"
83            complexFeatureInstance=ComplexFeatureInstance );
84
85  FeatureInstanceValue :
86    featureInstanceValue=Value |  complexFeatureInstance=
          ComplexFeatureInstance ;
87
88
89  ComplexFeatureInstance :
90    type=[Type] ( "(" refId=RefId ")")?
91    "{" ( (featureInstances+=FeatureInstance)
92        ("," featureInstances+=FeatureInstance)* )? "}";
93    Metamodel : name=ID ":" repository=ID   "from"  id=STRING (
94        "annotates" (annotates="ModelElements" |
95        annotates="Model") )?";";
96
97  Model :
98    name=ID ":" metamodel=[Metamodel] "from" file=STRING ";";
99
100 Type :
101   name=TypeName ":" metamodel=[Metamodel] (
102       isAnnotationProperty?="isAnnotationProperty")? ";";
103
104 String RefId :
105   STRING;
```