User Assistance during Domain-specific Language Design

Marco Kuhrmann
Technische Universität München, Institut für Informatik – IV
Boltzmannstr. 3
85748, Garching, Germany
kuhrmann@in.tum.de

ABSTRACT

Today, modeling is widely accepted technique in Software Engineering (SE). Nevertheless, the creation of modeling tools is a challenge. Supporting SE tasks by tools requires a lot of effort regarding e.g., the definition of data models, and methodological support. Even the standardized UML-notation requires a lot of work for being tool supported, because it has to be interpreted according to the domain of application, and the tools need to be programmed. Domain-specific languages (DSL) propose more efficiency: They provide exactly the modeling features required by the domain. Since DSLs are limited to a particular scope they need to be defined specifically for the considered domain. This is a time-consuming task that requires a lot of knowledge in (modeling) language design, user assistance, and tool support. In this paper, we discuss the need for extensive support for language engineers. We show first steps to assist users during the definition of visualization models for DSLs. We then motivate the extension of our *Process Development Environment* (PDE) platform to allow for a free-form-like, cooperative language design. We discuss this approach with respect to rapid modeling language creation, tool generation, and give examples from ongoing research.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: Programmer workbench; D.2.6 [**Programming Environments**]: Graphical environments, Integrated environments; D.2.10 [**Design**]: Representation

General Terms

Documentation, Design, Languages.

Keywords

Domain-specific Languages, Modeling Tools, User Assistance.

1. INTRODUCTION

It was at a conference on object-oriented modeling and development, where a modeling tool vendor stated: “Nobody wants to perform ‘real’ modeling, but only drawing pictures…”. Thus, although modeling seems to be a widely accepted methodology during software design and development, this statement motivates to re-think modeling and its application in projects. Looking at the past, for a while even the UML [13] was recognized as a sophisticated drawing tool (what we can still observe this if thinking about Microsoft Visio or the Omni Group’s OmniGraffle). During the last years modeling tools grew to comprehensive tool sets: UML became a standardized modeling language and notation. While UML is a “general-purpose” modeling approach that addresses a variety of modeling scenarios, specialized modeling techniques for certain domains were developed, e.g. Aris [2] for business processes, or Focus [18] for embedded systems. Even UML was customized according to certain domains, e.g., BPMN [10] for business processes, or SPEM [12] for software and system development processes, each accompanied by at least one modeling tool.

A major problem is, to our understanding, the complexity of the general-purpose modeling approaches, and in consequence the complexity of the associated modeling tools (e.g., Magicdraw UML [14] or the Enterprise Architect [19]). We can observe, that users need to be trained according to (1) the modeling methodology – which is often tool-specific – and also (2) to the usage of the tools themselves. Using modeling in a *model-driven development* (MDD) approach also requires to adjust the software development process itself, e.g., by considering generated source code and its handling. Especially in the domain of business information systems this could be the cause of not seamlessly applying MDD, but to use modeling rather to communicate in the team.

To make use of modeling in general, *domain-specific languages* (DSL) promise to offer the “best of both worlds”: (1) easy communication by using well-known and accepted domain objects in an appropriate notation, and (2) precision that allows for further processing of the domain models, e.g., to generate code, data models, and so on.

The problem here is: A DSL is designed according to specific domain requirements and, therefore, a concrete DSL is difficult to apply in other environments than the original planned one. Taking into account, the development of a DSL is also a (development) project; resources (time, budget, etc.) are consumed. A “throw-away” DSL for just one project therefore does not hold under economic considerations as long as *rapid language development* – similar to rapid prototyping – is not well supported for DSL development.

Problem Statement. The paper at hand considers the requirements of rapid language development. Providing analysts, designers, and other project roles with appropriate modeling languages and tools beyond standard solutions, which need to be interpreted and costly tailored, is a challenging undertaking. While DSLs offer a solution, their development is time-consuming work. To develop the DSL behind a generated end-user tool is hard and requires deep (conceptual and technical) knowledge of current

DSL development environments, e.g. Eclipse EMF/GMF, MetaCase, or the Visual Studio DSL tools. This makes *pragmatic* DSL-development tedious and uneconomic.

Contribution. Based on standard DSL tool kits we discuss how to ease the language development process for DSLs. We argue for extensive assistance for the DSL (language) engineer. At first we present a concrete assistant that shows what simple and easy support could look like. Second we sketch an extension to our DSL development platform PDE that will support free-form design of DSLs.

Outline. The remainder of the paper is organized as follows: In Sect. 2 we discuss related work, especially with regards to DSL-modeling and corresponding tools. We outline some issues with the currently available approaches from our experiences and motivate the need for extensive assistance. In Sect. 3 we shortly describe PDE. In Sect. 4 we present a concrete solution that supports DSL language engineers in defining visualization models. We also present some thoughts according to an extended language-modeling environment that supports easy visual DSL design. We conclude the paper in Sect. 5 and formulate the need for further research tasks.

2. RELATED WORK & DSL TOOLS

The development of modeling languages and modeling environments is a strongly discussed topic. We can roughly distinguish between the general-purpose approach UML [13], specific techniques for certain domains, i.e. Focus [18], and domain-specific languages [5], [8] somewhere in between. Thus, since many contributions in several areas of modeling are available we can hardly cover all aspects. Therefore we focus on current tools for meta-modeling and especially for DSL-design.

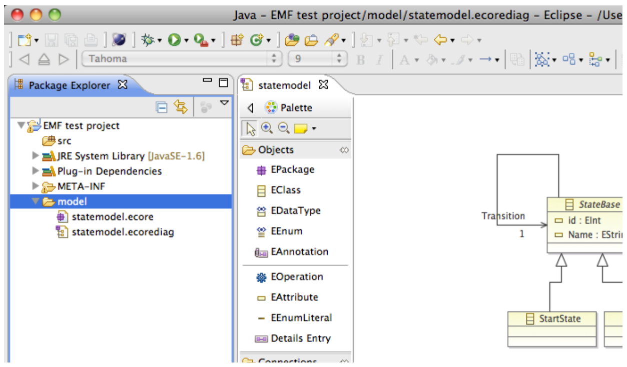


Figure 1. Sample DSL definition in the Eclipse environment

Metamodels can be discussed from different perspectives. Some good definitions in the context of language design can be found e.g., in [1]. We understand a metamodel as a formalism to describe (domain-specific) languages. To ease usage and understanding we skip mathematical definitions for now but focus on concrete *representations* of metamodels that are relevant when thinking about modeling tools.

Considering for example the Eclipse-based language modeling tools [4], [3] metamodels are represented by so-called ECore models, which are based on the OMGs MOF hierarchy [11]. The definition of a metamodel (DSL) is done using an UML-like notation subset as shown in Figure 1. EMF provides rich support for the definition of metamodels, which is shown by many concrete EMF-based languages (e.g., [20], or the variety of samples that can be found at [16]). The support is important for language engineers to adjust all aspects of a modeling language (structure and semantics). In fact, when capturing a domain is the current task, many of the powerful features are not required. We can state the same for the Microsoft DSL tool kit [1], [6], which we intensively used to develop PDE [9], [17] (see Sect. 3). The DSL tools are not based on UML but also use a structured, XML-based approach to define data models and add semantics using source code afterwards.

From our experiences we learned that the design of a DSL needs support in at least two areas: (1) to provide language "end users" with a modeling tool that supports them in handling concrete model instances and (2) to assist language engineers during the definition of a DSL. Almost all DSL tools address the first aspect. Eclipse for instance supports textual as well as visual DSLs and provides corresponding Eclipse-integrated editors. With our work on PDE we applied an alternative approach where the DSL is merged with a stand-alone editor framework during its transformations. So end users are provided with a stand-alone modeling tool according to their needs [9]. The second aspect is only partially addressed. Still, Eclipse, MetaEdit [16], and Visual Studio provide comprehensive support for the language engineers, if the domain of action is known and analyzed. If the language engineer should capture a domain and derive a DSL, no adequate support is given – especially during a domain analysis workshop, which is done with the stakeholders. For PDE we showed how to provide a modeling tool that also non-technophile stakeholders can use and understand. A similar support for the language creation process is currently not available.

3. PDE SUMMARIZED

To support our research we developed the *Process Development Environment* (PDE). PDE provides an infrastructure for the design of process languages and process authoring. The core functionality is based on the Microsoft DSL Tool Kit [1]. PDE adds several features, such as: Model visualization, metamodel modularization, or hotspots for validation functions.

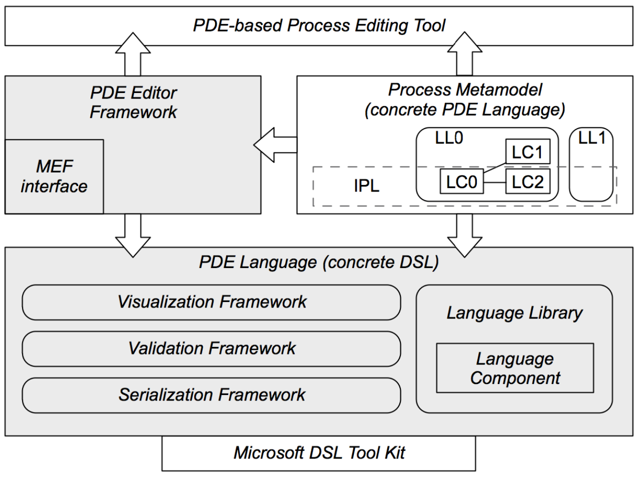


Figure 2 Architecture Overview of PDE

Figure 2 shows the architecture of PDE. The framework consists of two parts: (1) the PDE language, which is the extension of the DSL tools, and (2) an editor framework that provides the basic features to edit a designed process model. A concrete process language (a process metamodel) is a DSL based on the PDE extension of the Microsoft DSL SDK. The *PDE Language* is the basis for the process metamodel, which is merged with the *PDE Editor Framework* into a concrete modeling tool for process engineers.

PDE is originally designed to ease the development of process *metamodels* and to provide process engineers with corresponding tools to edit the resulting process *models*. Currently we work on extensions for PDE to enhance and support DSL development in general (as described in the paper at hands).

4. ASSISTANCE FOR DSL DESIGN

The design of a domain-specific language (DSL) is a hard and tedious task. Especially if considering visual languages, the DSLs grow complex, as the language not only contains structure and semantics but also information according to visual representations that build the graphical notation of the language. With EMF/GMF or the Microsoft DSL tools, platforms are available that support the creation of user defined (at most external [5]) DSLs. From our experience we know (at least) two topics that are hardly addressed: (1) the definition of so-called *visualization models* for the graphical notation, and (2) the definition of the language itself.

In this section we present at first a solution that provides language engineers with assistance to define visualization models. The second challenge we want to address with a draft of a concept that allows for a free-form-like language design approach.

4.1 Defining Visualization Models

A visualization model is an integrative part of a (visual) DSL that (1) defines the graphical notation of model elements, and provides (2) additional views to present certain model aspects to the users of the final DSL-based modeling tool.

For both aspects we presented examples in [9] and argued for their necessity with respect to users of the modeling tools. We do not want to explain the raw process of adding visualization to a Visual Studio-based DSL, which is a quite hard task. For PDE, which is explicitly designed to support visual languages, we developed a solution ourselves that eases this process.

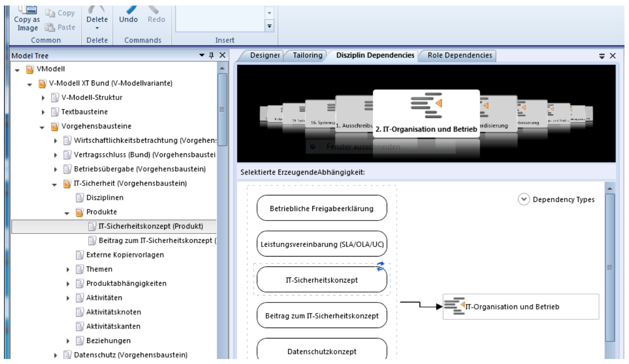
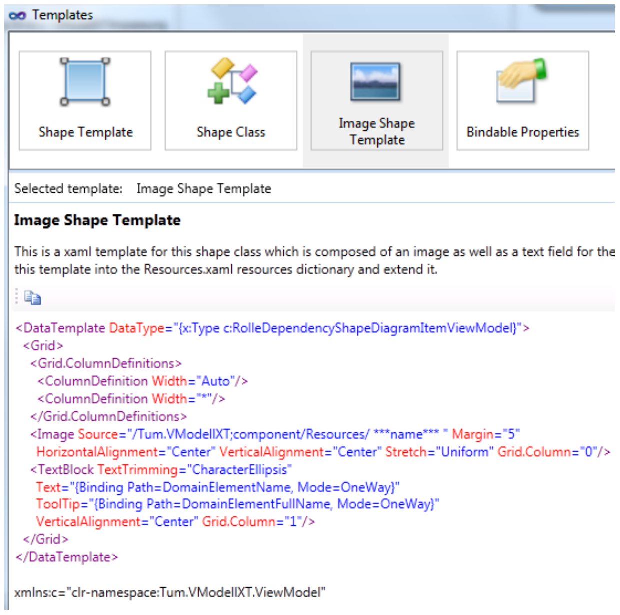

Copy as Image
Copy
Paste
Delete
Undo
Redo
Common
Delete
Commands
Insert
Model Tree
VModell
V-Modell XT Bund (V-Modellvariante)
V-Modell-Struktur
Textbausteine
Vorgehensbausteine
Wirtschaftlichkeitsbetrachtung (Vorgehen
Vertragsschluss (Bund) (Vorgehensbaustei
Betriebsübergabe (Vorgehensbaustein)
IT-Sicherheit (Vorgehensbaustein)
Disziplinen
Produkte
IT-Sicherheitskonzept (Produkt)
Beitrag zum IT-Sicherheitskonzept (
Externe Kopiervorlagen
Themen
Produktabhängigkeiten
Aktivitäten
Aktivitätsknoten
Aktivitätskanten
Beziehungen
Datenschutz (Vorgehensbaustein)
Designer
Tailoring
Disziplin Dependencies
Role Dependencies
2. IT-Organisation und Betrieb
Selektierte ErzeugendeAbhängigkeit:
Betriebliche Freigabeerklärung
Leistungsvereinbarung (SLA/OLA/UC)
IT-Sicherheitskonzept
Beitrag zum IT-Sicherheitskonzept
Datenschutzkonzept
Dependency Types
IT-Organisation und Betrieb

Figure 3. User-defined view that allows for filtered navigation

Figure 3 shows such a complex visualization. To define such a visualization by "hand" the language engineer needs to create certain *shape classes*, *WPF templates* and so on. The *PDE Language* designer contains an assistant that evaluates the DSL and provides the language engineer with customized snippets and class templates (Figure 4).

The language engineer only needs to switch the designer view in the PDE Language designer and to insert visualizations elements from the templates (he also can adjust them afterwards). To add and customize a comprehensive view as shown in Figure 3 takes usually less than 10 minutes, using the assistant from Figure 4.



Selected template: Image Shape Template

Image Shape Template

This is a xaml template for this shape class which is composed of an image as well as a text field for the this template into the Resources.xaml resources dictionary and extend it.

```
<DataTemplate DataType="{x:Type c:RolleDependencyShapeDiagramItemViewModel}">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto"/>
      <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Image Source="/Tum.VModellXT;component/Resources/ ***name*** " Margin="5"
     HorizontalAlignment="Center" VerticalAlignment="Center" Stretch="Uniform" Grid.Column="0"/>
    <TextBlock TextTrimming="CharacterEllipsis"
     Text="{Binding Path=DomainElementName, Mode=OneWay}"
     ToolTip="{Binding Path=DomainElementFullName, Mode=OneWay}"
     VerticalAlignment="Center" Grid.Column="1"/>
  </Grid>
</DataTemplate>

xmlns:c="clr-namespace:Tum.VModellXT.ViewModel"
```

Figure 4. Assistant to insert visualization model elements

4.2 Free-Form DSL Design

The experiences with assistants like the one shown above proved that an extended support dramatically eases the language definition. Hence, the assistant is still integrated with the Visual Studio-based PDE Language designer its application requires some technical understanding. The language designer component is far away from being easy to understand to stakeholders and supports the definition of a DSL on a fairly technical level.

Approach summarized. In this section we want to sketch PDE's further development with regards to a platform that not only allows for creating comprehensive modeling tools but also eases the language creation process itself. The language creation should be that easy that most of the "development" could take place cooperatively in stakeholder workshops. Therefore we sketch the idea of *cooperative language design* that allows for coupling creative and formal tasks during the definition of a DSL-based modeling language. The outcome of a cooperative language design is a modeling tool that (1) allows for free-form modeling with (2) respect to formal constraints defined by a domain-specific language.

In terms of creativity we consider tasks from domain analysis often performed in stakeholder workshops. The goal is to define a particular domain and name all domain elements and relations of importance (as well static as dynamic elements, e.g., artifacts or processes). Formal tasks contain the language definition itself and, in consequence, the generation of modeling tools. Language engineers usually perform formal tasks without the stakeholders' participation. The goal is to provide a (modeling) language that represents the domain under consideration and an appropriate tool to build models during workshops.

Idea detailed. In a cooperative modeling approach creative and formal tasks overlap to a certain point. We describe the idea referring to Figure 5: A stakeholder workshop to understand and capture the domain is done "as usual" – but instead of using a

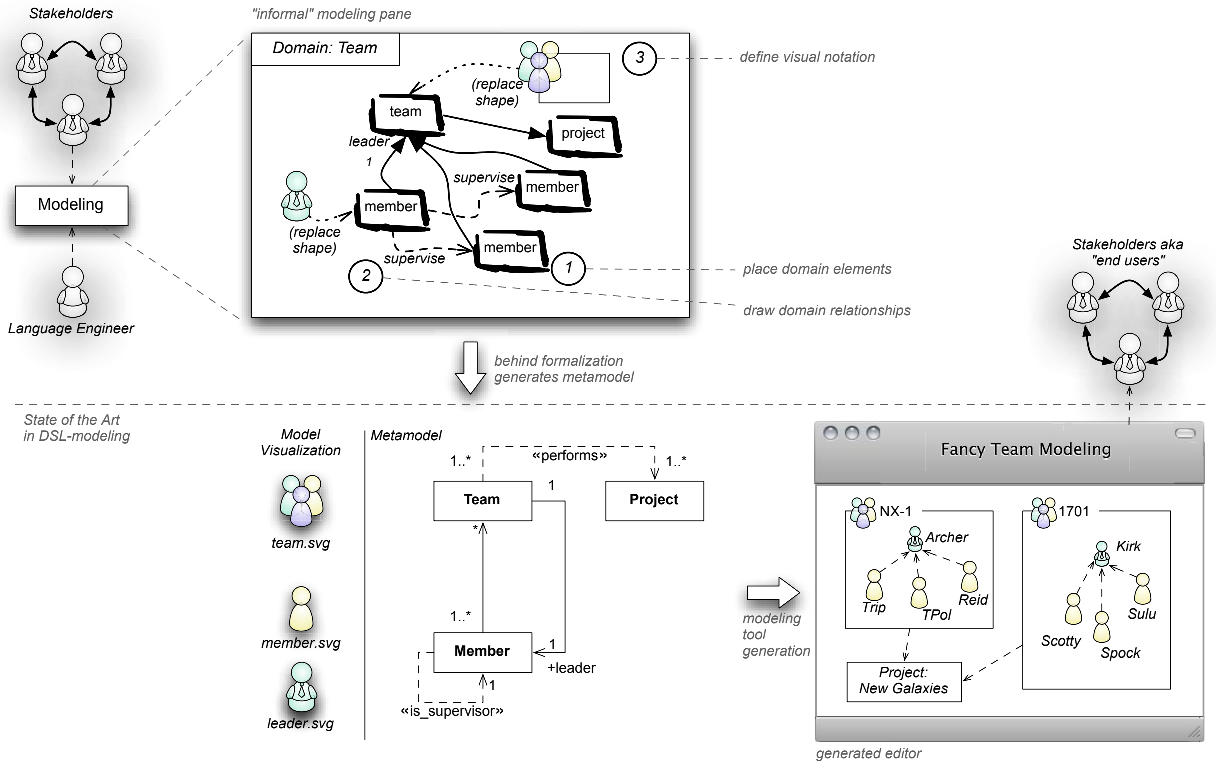


Figure 5. Cooperation language modeling approach (draft)

classical whiteboard a digital "informal" modeling pane is used. This pane collects domain entities, which are represented visually, and simple associations. In the workshop, entities can be collected, structured, combined, and so on. The goal is to express the domain using prototypical model instances as representatives for the domain under consideration. Stakeholders describe the domain as it is seen by their experiences.

Behind the modeling pane, a language-modeling tool captures the model prototype and translates into, or derives domain-specific language constructs to prepare the metamodel definition. The metamodel is mostly the result from the informal design and builds the basis to create the new DSL. Having the DSL, various tools, i.e., modeling tools, can be easily created. The stakeholders use the resulting modeling tools. The style of modeling, the notation and the semantics comply with the drafts made during the language creation workshops.

Current DSL-Design. Figure 5 sketches the process of language creation and modeling tool generation, and shows in the lower parts, which steps can be performed by existing concepts and technologies (see also Sect. 2). In [9] we gave an insight of developing modeling tools by using DSLs. We showed, how metamodels can be combined with visualization models, and can also be embedded into comprehensive tools.

These features we assume as state of the art. The addition with free-form language design requires further research, which we cover again in Sect. 5.

Limitations & Challenges. We have already done the first steps according to support language engineers. The approach sketched above goes beyond the current results and requires additional work. Thus, although we are still in the phase of conceptualization, we are aware of some limits and/or challenges:

- *Language Derivation*: The translation of the free-from design into a DSL implies the derivation of concrete language elements from the drawings. A 1:1 mapping, i.e., one drawing shape → one domain entity seems to be quite simple, but including e.g., abstraction or subtyping requires techniques to handle complex type systems, patterns and so on (i.e., graph transformations).
- *Structures and composition*: Drawings on the pane are at first "flat". If elements should be structured i.e., using containers, the corresponding language elements need to be automatically generated (see above).
- *Semantics*: A picture on the pane that is translated into a DSL needs structure as well as semantics. Semantics is quite hard to define and even harder to generate as it needs to be "drawn" somehow.
- *Language Complexity*: A DSL can be very complex. It might be hard or impossible to capture a complete domain using such an approach. Some kind of modularization needs to be used to describe small aspects, and afterwards to compose complete languages (i.e., language libraries as used in PDE).

Another aspect that needs to be considered is: *Is the modeling pane just another DSL?* If yes, is it possible to define a "meta-meta" language that allows for the definition of certain user-defined DSLs – and if so, what are the differences between that particular language and concepts already known from MOF? Also of importance is the question, if the "meta-meta" language has limitations itself and to what extend a user-defined DSL can be derived automatically?

Potentials. Nevertheless the approach promises to fasten DSL development. Even small DSLs for specific projects can be created very fast and for the most parts automatically. A "low hanging fruit" is the simplification of defining visualization models. With PDE we showed that rapid prototyping approaches could be applied to the development of complex models, too. The approach sketched above should transfer those concepts of prototyping to language development (meta-modeling) and serves to establish fast feedback cycles according to the quality of domain models.

5. CONCLUSION & FURTHER WORK

The paper at hand presents an example of the simplification of the development of visual DSLs. We presented a concrete solution that supports language engineers during the definition of visualization models to define a graphical notation for DSLs (Sect. 4.1). We further argued for extensive support for language engineers.

The design and implementation of a DSL is a tedious and challenging task that combines creativity and formal aspects. While the formal aspects are well supported by the currently available DSL tools (Sect. 2), creativity is hardly covered. The approach we sketched in Sect. 4.2 targets to the integration of domain analysis and language definition. We also presented some thoughts related to the challenges of such an approach. In fact, the creation of a domain-specific language is a task that aims to map certain domain concepts to formal languages. That particular mapping is currently done by the (human) language engineer and requires knowledge of the domain in advance.

We just sketched the idea of the approach but still plan to make further steps towards simplified DSL creation. A first concrete step is the further evaluation of the PDE platform. Currently we have several "real" modeling tools generated from that platform mainly targeting the domain of software development process models (in cooperation with our partners). We continuously extend the number of models, currently in the domain of modeling (common) artifacts. Thus, artifact-orientation is a very common concept that can be applied to many domains [15] we hope to be able to validate some of the thoughts presented above. Especially artifact representations that are close to data models are of interest. Such representations can be used as structural components for concrete languages.

6. ACKNOWLEDGMENTS

We want to thank Eugen Wachtel for his essential support during the development of PDE. We also thank Georg Kalus and Daniel Mendéz Fernández for the evaluation of PDE in their work.

7. REFERENCES

[1] S. Cook, G. Jones, S. Kent, and A. C. Wills. *Domain-Specific Development with Visual Studio DSL Tools*. Addison-Wesley Longman, 2007.

[2] R. Davis. ARIS Design Platform: Advanced Process Modelling and Administration. Springer, 2010.

[3] Eclipse Foundation. Eclipse Modeling Framework, 2010. URL http://www.eclipse.org/emf.

[4] Eclipse Foundation. Eclipse Project, 2010. URL http://www.eclipse.org.

[5] A. Fowler. *Domain Specific Languages*. Addison-Wesley, 2010.

[6] J. Greenfield, K. Short, S. Cook, S. Kent and J. Crupi. *Software Factories.* John Wiley & Sons, 2004.

[7] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler and S. Völkel. Design Guidelines for Domain Specific Languages. In *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM' 09)*, 2009.

[8] A. Kleppe. *Software Language Engineering*. Addison-Wesley, 2008.

[9] M. Kuhrmann, G. Kalus, E. Wachtel and M. Broy. Visual Process Model Design using Domain-specific Languages. In *Proceedings of SPLASH Workshop on Flexible Modeling Tools (FlexiTools)*, 2010.

[10] OMG. Business Process Model and Notation Version 2.0 (Beta 2), May 2010, URL http://www.omg.org/spec/BPMN/

[11] OMG. Meta Object Facility Version 2.0 (2006-01-01), January 2006. URL http://www.omg.org/spec/MOF/2.0.

[12] OMG. Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0. Technical Report, 2008.

[13] OMG. Unifed Modeling Language: Superstructure Version 2.2 (2009-02-02), February 2009. URL http://www.omg.org/spec/UML/2.2.

[14] Magicdraw UML. Company's home page, 2011. URL http://www.magicdraw.com/

[15] D. Mendez-Fernandez, B. Penzenstadler, M. Kuhrmann, and M. Broy. A Meta Model for Artefact-Orientation: Fundamentals and Lessons Learned in Requirements Engineering. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (MODELS 2010)*, 2010.

[16] MetaCase. Company's home page and samples, 2011. URL http://www.metacase.com

[17] PDE. The Process Development Environment, 2010. URL http://pde.codeplex.com

[18] B. Schätz. *The ODL Operation Definition Language and the Autofocus/Quest Application Framewoek AQUA*. Technical Report, Technische Universität München, 2001.

[19] Sparx Systems. Enterprise Architect. Company's home page, 2011. URL http://www.sparxsystems.de/

[20] B. Volz, and S. Jablonski. OMME – A Flexible Modeling Environment. In *Proceedings of SPLASH Workshop on Flexible Modeling Tools (FlexiTools)*, 2010.