A Framework for the Evaluation of Meta-Modelling Tools

Lutz Kirchner and Jürgen Jung
University Duisburg-Essen, Essen, Germany
lutz.kirchner@uni-duisburg-essen.de
juergen.jung@uni-duisburg-essen.de

Abstract: Meta-modelling tools are an important prerequisite for the utilisation of domain specific languages. They allow for the tool-supported definition of modelling languages and can be used to generate specific editors for the application of a language. However, due to the variety of tools on the market and their inherent complexity, it is a difficult task to select a tool, which suits a set of given requirements best. In this paper we suggest a framework, which comprises a set of criteria for the evaluation of meta-modelling tools. The framework can be used to assist the tool selection process and hence help to minimise the risk of choosing an inadequate tool for a given application scenario.

Keywords: Meta-modelling, meta-modelling tools, tool evaluation, domain specific languages

1. Introduction

Modelling is of increasing importance in the context of the development and management of information systems. *Models* are used for the specification of software as well as the description of business processes. In addition, they serve as a basis for software development and workflow management. The syntactical rules for creating a model are defined by a *modelling language*. Popular modelling languages are the Unified Modelling Language (UML), the Entity-Relationship-Model (ERM), and Event-driven Process Chains (EPC). Modelling languages can be specified in various ways. For a large variety of potential model applications, domain specific languages (DSLs) are of increasing relevance. They are able to fill the gap of applicability, which is left open by general-purpose languages (GPLs), like those mentioned above, due to the DSL´s semantically richer concepts. DSLs are specifically tailored for one domain, which comes with several advantages (see paragraph 2). A predominating approach for specifying (graphical) modelling languages, like e.g. DSLs, is based on meta-models, which are in fact models of modelling languages.

Tool support is a paramount prerequisite for an efficient modelling process. Due to the literally infinite number of possible DSLs, it is uneconomical to create a specialised modelling tool for every language. Hence, the usage of a meta-modelling tool, which can provide tool support for the definition and application of a DSL, seems to be an attractive option. There are a number of tools on the market, which differ substantially in their approach to meta-modelling as well as their capabilities and features. Since meta-modelling tools in general are of significant complexity, it is hard to determine which tool is suitable for a given application scenario. Hence, a methodical approach to the evaluation of candidate tools in form of a conceptual framework is desirable.

This paper introduces a framework for the evaluation of meta-modelling tools. It comprises a set of categorised criteria, which serve to identify and document mandatory and optional features of the tools. We applied this framework to a number of meta-modelling tools, which we plan to use for the realisation of special tools concerning the modelling of IT-landscapes and resources in general. However, before introducing the framework, we want to briefly outline the notions *meta-modelling* and *meta-modelling tool* as well as some fundamental differences between *domain specific languages* and *general purpose languages* (GPLs) in the next paragraph. Following, we discuss the evaluation framework and its set of criteria. Additionally, we suggest a generic evaluation process. The framework will be exemplarily applied to two concrete tools, before we conclude by summarising the gained insights and give an outlook to future research.

2. Terminology

The notion *meta-modelling* describes the act of creating a model of a modelling language, thus defining its abstract syntax and semantics (cf. Geisler, R., Klar, M. and Pons, C. 1998 and Gitzel, R. and Hildenbrand, T. 2005). Furthermore, we regard a *meta-modelling tool* primarily as a modelling tool that allows for performing meta-modelling, i.e. defining the abstract syntax and – to a lesser degree – the semantics of a modelling language. Additionally such a tool has to provide facilities to define a concrete syntax (i.e. notation), and allow for the creation of user models. This definition is intentionally kept broad, but nonetheless it provides us with an adequate guideline for the selection of candidate tools for the evaluation. The modelling languages mentioned above (UML, ERM, EPC) can be used for the depiction of any

Reference this paper as:
Kirchner L and Jung J (2007) “A Framework for the Evaluation of Meta-Modelling Tools” *The Electronic Journal Information Systems Evaluation* Volume 10 Issue 1, pp 65 - 72, available online at www.ejise.com

kind of domain. These languages are called *general purpose languages* since they are largely domain independent due to the provision of generic concepts on a semantically low level. By contrast, *domain-specific languages* contain language features, which are suitable only for one specific domain and hence can comprise semantically richer concepts than GPLs. Examples for DSLs are languages for modelling the firmware of mobile phones, a negotiation protocol or IT-architectures. A number of reasons, why DSLs potentially provide significant advantages compared to GPLs can be found in various publications (e.g. in (Esser and Jannek 2001) and (Luoma, Kelly and Tolvanen 2004), among others). A DSL usually provides application related concepts, which a GPL does not (e.g. a concept *Computer* instead of the generic concept *Class* in the context of the modelling of IT-resources). The use of such domain specific concepts can decrease the number of errors in a model and thus increasing model quality, since their semantic correspondence with the depicted real world concepts is usually straight forward. Hence, a user model will be more vivid in many cases and easier to comprehend for a domain expert. Additionally, a meta-model of a DSL includes rules, which regulate the use of model elements according to the specifics of a domain. This holds e.g. for the creation of relations between certain model elements. Such rules can be enforced by a modelling interface, which was created by a meta-modelling tool on the basis of DSL´s meta-model. Another benefit results from a more specific and therefore effective code generation from models. Due to their limited applicability, DSLs are usually not supported by conventional modelling tools. The lack of tool-support can be leveraged by the use of a meta-modelling tool.

3. Evaluation framework

The suggested framework for the evaluation of meta-modelling tools consists of two top level categories of evaluation criteria, whereas the second category includes three subcategories:

- General Evaluation Criteria for Software Tools
- Evaluation Criteria for Meta-Modelling Tools
    - Tool Architecture
    - Modelling Language Specification
    - Modelling Language Application

In the following paragraphs the criteria, which belong to the respective categories, will be discussed.

3.1 General evaluation criteria for software tools

This category contains criteria, which are of interest for the projected purchase of almost any kind of software tool, regardless of its area of application. Due to their general nature, we discuss these criteria very brief in this paragraph.

Cost related: costs, hardware requirements and variants. Typical costs related to software are related to its acquisition and operation. The latter includes primarily costs for licenses and support (updates, hotline). The hardware, which software requires to be executed, is also responsible for the generation of costs. The higher the hardware requirements are, the more expensive the according hardware components become. Furthermore, a tool can be available in several variants, which differ in terms of their range of functionalities. The purchase of optional features often results in a higher license price. Hence, it has to be evaluated, whether a certain functionality is required.

Usability: ergonomics and documentation. Ergonomics addresses the compliance of a tool with software ergonomic standards. These describe guidelines for the design of menus and dialogs, the information presentation, and the user interface in general. An example for such a standard is the *ISO 9241 Usability Standard*. It deals with various aspects of software ergonomics in paragraphs 10—17. If the software complies with such standards, the time and effort involved in learning to work with a tool can be significantly reduced. In a similar way, the quality of the available documentation has an effect on the time, which is needed to familiarise with a tool and utilise its full potential. Possible sources of documentation are manuals, online and context sensitive help functions, hosted forums, hotlines, among others. These documents can be available in various languages.

(Un)Installation. Concluding the general criteria, the installation and uninstallation process of a tool has to be examined. The installation should be scalable according to the user's needs, whereas the uninstallation should be complete and clean.

3.2 Evaluation criteria for meta-modelling tools

The evaluation criteria in this paragraph deal with aspects, which are of particular interest for meta-modelling tools. The category is divided into three subcategories, which serve to analyse a tool's architecture and cover aspects relevant for the specification and application of modelling languages.

3.2.1 Tool architecture

The overall architecture of a meta-modelling tool is a basic factor, which determines its performance and flexibility. The following criteria serve to evaluate a tool's architecture.

General architecture: modularity. It is difficult to determine which concrete architecture is the most advantageous for a meta-modelling tool. In general, a modular tool design is expected to provide a higher flexibility compared to monolithic approaches. This holds especially with respect to future updates from the tool vendor and possible user modifications. A tool should provide modules responsible for storing information about the meta-meta-model, meta-models and models as well as for providing model persistency (database and persistency mechanisms), a model viewer and a model builder. Due to tight integration of the models on the various abstraction layers, all models are typically managed in one module. An example for such a modular architecture is described in (Karagianis and Kühn 2002). The authors introduce a generic component-based architecture for meta-modelling tools, which origins from similar requirements.

Model management: model storage mechanism and configuration management. Regarding model management, a configuration management for the meta-models and user models should be implemented. The management of versions and variants of models, as well as a rollback function are desirable. Thus, transparency regarding the project progress is provided and the redoing of certain work steps as well as the starting over from older versions possible. Strategies for the solution of version conflicts have to be implemented as well (automated as well as dialog-based). As a strategy for model storage and therefore a basis for configuration management, a repository which provides built-in security and transaction mechanisms seems advantageous. Collaborative work on a project will largely depend on the availability of these features.

User defined extensibility. User defined functions can enhance the usability of a modelling tool. These functions may be added by using internal languages or addressing the tool's external application programming interface (API). Complemented functions can range from user implemented model analyses to the modification of the user interface, among others. An onboard language provided by the tool can offer the necessary expressive power for implementing the desired features. If none is provided, the API has to be used. However, it has to be possible to access all model-specific data and model elements through the used interface, from both model- and meta-level. Beside the power of the extensibility features, the simplicity of using them should be reviewed. Some tools provide extension mechanisms, which require the use of proprietary, sometimes little documented scripting languages for the implementation of user functions, whereas others can be programmed using standard languages like Java. The latter usually reduces the familiarisation time, presumed that basic knowledge of the programming language is existent.

Tool integration. Another criterion of this category deals with the means of integration with other tools, a meta-modelling tool may provide. On a technical level, the integration with modelling tools can be achieved by supporting exchange formats (e.g. XMI) or providing interfaces in order to exchange models transparently to the user (Schloegel, Oglesby and Engstrom 2002). This can be accomplished by implementing plug-ins, using the API of external tools. However, beside the exchange format, an important aspect for the integration is the compatibility of the modelling languages or meta-modelling languages respectively, since without it a successful exchange of models cannot be achieved. Additionally, integration with office applications or external editors can be useful for including documentations in or running programs from a model.

3.2.2 Modelling language specification

This set of criteria focuses on the evaluation of concepts, which are related to language specification tasks.

General approach to meta-modelling. There exist different approaches to the definition of the syntax and semantics of a modelling language (Esser and Jannek 2001; Hofstede and Proper 1998; Luoma, Kelly and Tolvanen 2004). E.g. a grammar-based approach (e.g. with *Extended Backus-Naur Form*; EBNF) can be used to describe a language's syntax in a formal and precise way. However, this tends to be a time consuming and complicated approach and thus is only recommended, if it is planned to base formal proofs on a model created with the resulting language. For defining visual modelling languages comprising graph-like structures, a meta-model-based approach is preferable. That approach is only semi-formal, but offers the advantage of avoiding a paradigm shift regarding the meta-model and its instances, compared to a strict formal definition (Frank 1999). This results in a more intuitive meta-modelling process and therefore tends to foster model quality. In this context a tool may allow for a varying number of

abstraction levels – or layers – of modelling. However, the provision of at least an explicit meta-model and a model layer is mandatory.

Definition of language concepts: syntax and semantics. The way a tool addresses the definition of the language concepts on the meta-layer is also an important evaluation aspect. First of all, a tool should differentiate between the abstract and concrete syntax of a concept. The abstract syntax (at least for graph-based languages) defines node and edge types with their respective properties. In this context, a tool should provide an automated check for the correctness of a meta-model against the background of the meta-meta-model. According to (Jung and Kirchner 2005) it should be possible to define constraints, which e.g. prevent circular relations or enforce bipartite graphs on model-level, among others. The concrete syntax of a language defines graphical representations of nodes, edges and various properties of model elements. Properties can be listed inside the symbol of a model element or be located externally. Further possible features are the definition of docking points for elements, different routings of edges, a different graphical representation of the same concept according to the context and user preference (e.g. a UML interface displayed as a circle or a rectangle), or the combination of edges to complex edges. The latter refers to edges, which connect more than 2 nodes like in organisational charts or the generalisation relationship in UML. Which representation options a tool should provide, strongly depends on the kind of modelling languages it is used to implement. However, in general it can be stated, that the more complex notations can be implemented, the more visual expressiveness a language may hold.

Complementing the syntax definition, a tool may provide means of defining the semantics of a modelling language. This can be achieved by mapping language concepts to an adequate modelling language or a common programming language (Hofstede and Proper 1998), which possesses concepts with well defined semantics (e.g. Petri nets). However, due to the complexity of the implementation of such concepts, significant tool support in this area can not be expected yet. Alternatively, a concept's semantics can be specified with an onboard scripting or a constraint language, like the OCL (*Object Constraint Language*), or by attaching natural language comments to the meta-model elements.

3.2.3 Modelling language application

The criteria introduced in this paragraph deal with aspects of the application of modelling languages.

Generation of modelling tools and predefined modelling languages. First, a tool should provide a language specific user interface with an editor including specially tailored menu items, which support an intuitive modelling process with a previously specified modelling language. Furthermore, the provision of a number of predefined, common GPLs (e.g. UML or ERM) would be desirable. If these languages are already adequately implemented, there is no need to redefine their meta-models or to purchase complementary tools. They can be used as provided, analysed (e.g. in a teaching scenario) or extended (e.g. UML with stereotypes or user profiles).

Model transformation. Since modelling often is an intermediate step in a process aiming at constructing information systems, it should be possible to transform a model into other representations. The transformation target can be other visual modelling languages or, more frequently, implementation-level documents. The latter (i.e. source code generation) would allow for the transformation of object models to a programming language (e.g. UML to Java) or workflow models to workflow schemata among others (Oberweis 1996). E.g. (Jung 2004) presents a meta-modelling tool-based approach for generating XPDL-compliant (*XML Process Definition Language*) workflow definitions from business process models. For analysis purposes the automated construction of models on the basis of source code may be supported, too. The rules for every model transformation should be user definable in a flexible way to ensure an appropriate mapping of one language to the other, e.g. using included scripting languages.

Simulation. The usability of a model can be significantly enhanced, if it can serve as a basis for simulation. Especially in the area of business process and workflow modelling, simulation is a pivotal application of a model (Jung 2003; Zelm 2003). Simulation can serve to analyse projected as well as existing domains for weak spots and bottlenecks, which decrease the overall performance of the modelled system. In (Nikoukaran, Hlupic and Pail 1998) a framework for the evaluation of simulation software is introduced, which in extracts can also be applied to evaluate the simulation capabilities of a meta-modelling tool. Particularly, the criteria for *input*, *execution* and *output* are of interest in our context. Input deals with the possibilities that a tool provides for assigning data relevant for simulation to model elements and for defining additional functions, which influence the simulation execution. The control of the execution should

include the definition of the number of execution steps, the simulation speed, stop/pause/resume functions, animated execution, among others. The output of the result should be available in textual as well as graphical form, and should be stored permanently in a database. This leads to the following short summarisation of basic simulation requirements, which a tool should fulfil:

- simulation input
  - Definition of simulation functions
  - Assignment of (instance-related) data to model elements
- Simulation execution
  - Control of execution parameters (steps, speed, general control)
- Simulation output
  - Various result representations (textual, graphical)
  - Permanent storage of results for future reference

Metrics. As an instrument for a basic measurement of a model's quality, metrics can be used. Generic metrics, e.g. for counting the number of instances of a type, may be predefined. Especially in the context of object-oriented modelling exists a large number of metrics (Henderson-Sellers 1996). Other metrics may be specified by the user with respect to the application to specific user defined modelling languages and models. Metrics can help to improve the quality of a model by providing an early warning mechanism, hinting at possible weak spots in a model. The implementation of their management should cover their creation, modification and storage.

Model documentation. To conclude the description of the framework, this criterion deals with the issues of documenting a model or meta-model respectively. For an online distribution of a model's documentation the report format should be navigable (hypertext-based). This can be implemented e.g. by PDF or HTML files. Additionally, popular document formats (like e.g. MS-Word DOC, plain text) may also be supported, since a user can easily include a report into his own documents. To generate an appropriately detailed documentation, the generator has to have full access to all relevant parts of a model. The resulting level of detail should be user definable and therefore adjustable to the intended application of the documentation. Further means of documentation can provide prints of models. Functions like scaling and tiling of models as well as a comfortable preview mode should be implemented.

4. Sample evaluation

To ensure a successful evaluation process, the application of the above introduced framework should be guided methodically. We suggest a rather simple process model which consists of the following steps:

- Determine the application scenario of the meta-modelling tool and make it explicit to all participants involved in the evaluation
- Determine the relevance of each evaluation criterion with respect to the application scenario
- Review every meta-modelling tool selected for the evaluation with respect to each criterion
- Assess the results and select the most appropriate tool

We illustrate the usage of this process model by an excerpt of the documentation of an evaluation process which we already conducted. In this context we introduce the two most promising meta-modelling tools: MetaEdit+ (http://www.metacase.com) and Cubetto (http://www.semture.de). These tools have been selected because on the one hand they were developed by commercial vendors and ready for the market, but on the other hand free of charge for our specific application. Other tools have been excluded since they are research prototypes (ConceptBase[1], H2 Toolset[2]) or not explicitly offered as meta-modelling tools (ADONIS[3], Metis[4]). The evaluation has been performed by two researchers who worked closely together during the application of the framework. However, its applicability in conjunction with a larger group of participants was not tested yet. In this case it may be necessary to supplement the above process model with additional activity steps, which regulate the coordination of the group's individuals to avoid communication problems and as a consequence inconsistencies in the evaluation results. In general, to guarantee an adequate level of quality, it is recommended that an evaluation is being conducted by persons who have at least basic knowledge of meta-modelling tools and some evaluation experience. Additionally, all participants have to share a common understanding of how to apply the evaluation framework.

Application scenario of the meta-modelling tool. A meta-modelling tool is required in order to develop a modelling tool for the application in the area of

[1] http://www-i5.informatik.rwth-aachen.de/CBdoc/
[2] http://www.wi.uni-muenster.de/improot/is/pub_imperia/doc/1771.pdf
[3] http://www.boc-eu.com/bochp.jsp?file=WP_582571cc1ed802de.46e381.f59775478f.-7f17
[4] http://www.troux.com/products/metis_architect/

IT-management. Several languages have to be implemented. First, a business process modelling language, which has been developed at the author's institution, is needed. Furthermore, a resource modelling language and a language for the modelling of IT-landscapes are to be implemented. The provision of predefined languages for the modelling of software systems (like UML and ERM) is considered relevant, whereas common process modelling languages (like EPC) are of less importance. The resulting modelling tools are planned to be used in lectures and tutorials as well as in research projects. License costs for the meta-modelling tools and runtime licenses for the generated modelling tool should be minimal (close to zero).

Relevance of evaluation criteria. The relevance of some of the criteria we used for the evaluation is presented in table 1. The first column lists the identifying number of each criterion and the second one its name. The third column contains a four scaled quantifier representing the relevance of the criterion (3: very important; 2: important; 1: less important; 0: unimportant). License costs are regarded as very important, because there exist no funds dedicated to the purchase of a tool. Please note, that a high relevance of costs does not imply a positive interpretation of higher costs. A complete uninstallation is regarded as unimportant because the selected tool is intended to be used for an extended time period. This of course only holds when a tool update does not require a previous uninstall of the older version (which we presume at this point). Predefined modelling languages are regarded as very important (their existence is mandatory). Nevertheless, this criterion has not a single quantifier but a quantifier for each language separately. There are languages which should be implemented (EPC are of high value) and others which are "nice to have" (PN). UML and ERM are mandatory languages. They both have to be implemented "ready to use" by the meta-modelling tool. Simulation is not regarded as very important because we do not plan to perform analyses on this basis in the near future. Nevertheless, simulation is an important instrument whose availability in future versions of a tool may become a required feature. Metrics represent a less important functionality. We are not planning to use them for the time being, but this also might change in the future.

Table 1: Relevance of selected criteria

Number	Name	Quantifier
1.1.1	License costs	3
1.3.2	Complete Uninstallation	0
2.3.1.2	Predefined Modelling Languages	
	Event-driven process chains (EPC)	2
	Petri nets (PN)	1
	Unified Modelling Language (UML)	3
	Entity-Relationship-Model (ERM)	3
2.3.3	Simulation	2
2.3.4	Metrics	1

Tool evaluation. The evaluation results of two tools are displayed in table 2. We used a rating scale ranging from 0 (does not fit our need at all) to 10 (perfectly fits our needs). MetaEdit+ is not free for academic users but there is a special discount of 90% of the list price. This has been rated as '5' because there are a number of free tools and only a few are with costs. Cubetto is free of charge and therefore rated with '10'. Both tools can be easily removed from the system by deleting there installation directories (10 points for each tool). MetaEdit+ only provides UML as a predefined language. Consequently EPC, PN and ERM are rated as 0. The recently specified UML 2.0 is supported except for two minor diagram types which we do not need at the moment. Cubetto fully supports EPC including relationships to other diagram types (e.g. organisational charts, ERM) but does not include PN. ERM are included but in the form of the rather unwieldy original notation by Chen. We usually use a simplified and more modern ERM notation. Hence, ERM in Cubetto is rated 5. The implementation of the UML does not conform to UML 2.0. Only 5 out of 13 diagram types are supported, and the model editor is rather flawed. Simulation is neither implemented in MetaEdit+ nor in Cubetto. Both tools do not directly support the modelling of instances and the animation of process models. Consequently, both tools are rated zero. The determination of metrics is not directly supported by MetaEdit+ and Cubetto. However, each offers a scripting language. The language of MetaEdit+ primarily supports report generation. Its features comprise the generation of text and the navigation over model elements. It does not support iterations (for- or while-loop) and is therefore not Turing-complete. Cubetto uses JavaScript as a scripting language – which is Turing-complete.

Nevertheless, all queries regarding metrics have to be implemented by the user.

Table 2: Tool Evaluation (Excerpt)

Number	MetaEdit+	Cubetto
1.1.1	5	10
1.3.2	10	10
2.3.1.2		
- EPC	0	10
- PN	0	0
- UML	10	3
- ERM	0	5
2.3.3	0	0
2.3.4	2	4

Result assessment and tool selection. The preliminary result of the evaluation is shown in table 3. The values in the fourth and the sixth column are computed by multiplying the tools rating by the quantifier. Cubetto scored 172 points and MetaEdit+ 165 (note, that the score includes results which are not listed in the tables). This result was taken into account during the decision process for one of the tools. With respect to the criteria in table 3, Cubetto has lower costs and includes more predefined modelling languages than MetaEdit. But, the implementations of ERM and UML are far from perfect. Hence, it would be necessary to re-implement these languages in order to meet our requirements. This and the better overall usability lead to our final decision for MetaEdit+.

Table 3: Preliminary evaluation result (Excerpt)

Number	Quantifier	MetaEdit+		Cubetto	
1.1.1	3	5	15	10	30
1.3.2	0	10	0	10	0
2.3.1.2					
- EPC	2	0	0	10	20
- PN	1	0	0	0	0
- UML	3	10	30	3	9
- ERM	3	0	0	5	15
2.3.3	2	0	0	0	0
2.3.4	1	2	2	4	4

5. Summary

In this paper we suggested the use of a framework for the evaluation of meta-modelling tools. After an introduction to the topic and a short discussion of meta-model related terminology in paragraphs 1 and 2, the framework is introduced in paragraph 3. It consists of general and meta-modelling tool specific criteria. An exemplary application of the framework is demonstrated in paragraph 4. The value of such an evaluation framework not only depends on the covered criteria, but also on the expertise and background of the evaluating person. The determination of the relevant criteria and their significance to the intended area of application, as well as the application to the candidate tools, usually requires some background in meta-modelling and at least basic knowledge of the characteristics of possible target domains. Hence, the process in paragraph 4 could be extended by a complementing first step: the determination of the group of persons, who are suited best for the evaluation. The criteria presented in this paper are by no means complete, but should nevertheless allow for selecting a tool, which fulfils the given requirements. However, future work includes the documentation of additional criteria. E.g. the support of meta-modelling patterns, or the compliance with a meta-modelling method may be examined. It is important to note, that the results, as they are displayed in tabular form in this paper, usually should not be used without any interpretation in order to reach a decision. The results for every single criterion should be considered. The quantifiers, which value the importance of a criterion, often mirror the subjective opinion of one ore more individuals. Hence, they have may be adapted according to the requirements and preferences of the user. Nonetheless, our framework proved to be very useful in the process of determining the most suitable meta-modelling tool for our planned application. It will be further developed and complemented in the future.

References

Esser, R. and Jannek, J.W. (2001) „A Framework for Defining Domain-Specific Visual Languages“, *Proceedings of the Workshop on Domain Specific Visual Languages at OOPSLA 2001*, Tampa Bay, pp. 111—124

Frank, U. (1998) *The MEMO Meta-Metamodel*, IWI Research Report No. 9, Koblenz: Institute of IS Research

Frank, U. (1999) *Memo: Visual Languages for Enterprise Modellin*g, IWI Research Report No. 18, Koblenz: Institute of IS Research

Geisler, R., Klar, M. and Pons, C. (1998) *Dimensions and Dichotomy in Metamodelling*. Research Report No. 95-8, Berlin: Institute for Communications and Software Technology

Gitzel, R. and Hildenbrand, T. (2005) *A Taxonomy of Metamodel Hierarchies*, Research Report, Mannheim: Department of IS

Henderson-Sellers, B. (1996) *Object-Oriented Metrics: Measures of Complexity*, Upper Saddle River: Prentice Hall

Hofstede, A.H.M. ter and Proper, H.A. (1998) „How to Formalise it? Formalisation Principles for Information Systems Development Methods“, *Information Systems and Software Technology*, Vol. 40, No. 10, pp. 519—540

Jung, J. and Kirchner, L. (2005) *Meta-Modelling Tasks – Prototypical Language Features*, White Paper, Essen: Information Systems and Enterprise Modelling

Jung, J. (2003) „Basic Conceptualisation of a Resource Modelling Language“, *Information Technology and Organisations – Conference Proceedings*, Information Resources Management Association International Conference, Philadelphia, pp. 827—831

Jung, J. (2004) *Mapping of Business Process Models to Workflow Schemata -- An Example Using MEMO-OrgML an XPDL*, IWI Research Report No. 47, Koblenz: IS Research Institute

Karagianis, D. and Kühn, H. (2002) „Metamodelling Platforms“, *Proceedings of the 3rd International Conference EC-Web 2002 - Dexa 2002*, Aix-en-Provence, pp. 182—196

Luoma, J., Kelly, S. and Tolvanen, J.-P. (2004) „Defining Domain-Specific Modelling Languages: Collected Experiences“, *Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modelling*, Vancouver, pp. 1—10

Nikoukaran, J., Hlupic, V. and Paul, R.J. (1998) „Criteria for Simulation Software Evaluation“, *Proceedings of the 1998 Winter Simulation Conferenc*e, Washington, pp. 399—406

Oberweis, A. (1996) *Modellierung und Ausführung von Workflows mit Petri-Netzen*, Stuttgart: Teubner

Schloegel, K., Oglesby, D. and Engstrom, E. (2002) “Towards Next Generation Metamodeling Tools“, *Proceedings of the Second Workshop on Domain-Specific Visual Languages – OOPSLA 2002*, Seattle

Zelm, M. (2003) „Resource Requirements of Enterprise Management“, in Bernus, P., Nemes, L. and Schmidt, G. (ed) *Handbook on Enterprise Architecture*, Berlin: Springer

