On the Implementation of Tools for Domain Specific Process Modelling

Stefan Jablonski, Bernhard Volz and Sebastian Dornstauder

University of Bayreuth, Bayreuth, Germany

Abstract. Business process modelling becomes more productive when modellers can use process modelling languages which optimally fit to the application domain. Domain specific modelling is the discipline that deals with the proliferation of domain specific modelling languages. The general tenor is that the more a modelling language fits to an application domain, the more efficient and effective an application can be modelled. In this paper we address the issue of providing domain specific languages in a systematic and structural way without having to implement modelling tools for each domain specific language separately. Our approach is based on a two dimensional meta modelling stack.

1 Introduction

"The only constant is change" is a common quotation in literature when business process management is characterized. Without anticipating the introduction of a modelling hierarchy, the phenomena of change can be classified according to the process modelling levels they occur. Starting at the "lowest" level, running process instances might have to be changed to react to a sudden shift in the application. Among others, [24], [3] and [22] are investigating this issue and suggest adequate solutions. Stepping one level up, the process model (definition) might have to be changed since it has become obvious that from now on a certain application will be performed in a different way [24] [9]. Nevertheless, it is possible to even step up another level hierarchy. Change on this level means to alter the modelling language we focus on in this paper. For process aware information systems this kind of change means an evolution of the whole system over time.

Why is the change of a process modelling language an issue that is worth investigating? One can argue that a process modelling language should always remain untouched. However, we fully comply with the interpretation of change as being related to diversity [4]. Although that book discusses change in the context of programming languages, we can transfer the results to process management. The authors of [4] notice that diverse domains will be characterized by diverse customer requirements. This observation can seamlessly be adopted in the business process management domain. Here, the programming language is the modelling language and the deployment platform corresponds to the process execution infrastructure.

Jablonski S., Volz B. and Dornstauder S. (2009).
On the Implementation of Tools for Domain Specific Process Modelling.
In *Proceedings of the 4th International Conference on Evaluation of Novel Approaches to Software Engineering - Evaluation of Novel Approaches to Software Engineering*, pages 109-120
DOI: 10.5220/0001861401090120

We fully subscribe to the argument of [4] that the right languages enable developers to be significantly more productive. Besides we agree with the requirement that "we need the ability to rapidly design and integrate semantically rich languages in a unified way". This means on the one hand that each domain may and finally has to create its individual, specific language (domain specific language, DSL). On the other hand it means that a common starting point for these language developments is assumed. It is important to sustain – despite the diversity of DSLs – a kind of comparability and compatibility between them. We finally agree that meta modelling provides capabilities to achieve this.

Changes of a modelling language need not to be huge. For example, in [16] process steps are tagged to indicate whether they are prohibitive or mandatory. Although being unspectacular, this tagging is very valuable for the execution and evaluation of a process model. Standard process modelling languages like BPMN [18] do not offer this special kind of tagging a priori.

At this point it has to be discussed whether changing a process modelling language is counterproductive since it diminishes the possibility to exchange process models with partners. Here, we assume that each development of a DSL takes a standard language (e.g. BPMN) as a starting point. The following two arguments support the idea of a domain specific process modelling and – therefore – the adaptation of a standard modelling language:

First, domain specific adaptations are decisively enhancing the applicability of a process model within that domain. Adaptations are almost exclusively of interest within a domain. Thus, it is favourable to support adaptations.

Second, the use of a standard modelling language especially pays off when process models have to be exchanged with partners. Using a meta modelling approach, it is easy to distinguish between modelling elements of the standard language and those of a domain specific adaptation. Thus domain specific adaptations can be filtered out before a process model is exchanged. Although filtered process models lose information they are relevant and readable for receiving partners since the latter merely contains standard modelling elements. Assuming that domain specific extensions are primarily of interest for the domain developing them, this loss of information is tolerable.

Building up on these assumptions we present a meta modelling approach which supports the definition of domain specific process modelling languages. The special feature of our approach is that DSLs are derived from a common basic language which most probable will be a sort of standard language. All language definitions will be based on a meta model. This strategy bears major advantages.

- All derived DSL share a common set of modelling constructs. Thus, they remain compatible and comparable to a certain extend.
- The definition of a DSL is performed in a systematic way by extending the meta model of such a language.
- Extensions made for one DSL could be inherited by other domains, i.e. DSLs, if it is considered to be valuable for the new domain as well. This feature supports reuse of modelling constructs greatly.
- Tools can be built that support different DSLs at the same time. It is not necessary to build a special tool for each DSL.

So, we deliberate on the benefit of a standard notation and of a customized one. We definitely favour customization – as argued in [4] – since productivity is supported decisively better. Nevertheless, data exchange is still feasible.

The focus of this paper lies on tool support for domain specific modelling languages. The foundations of a domain specific processes modelling tool are discussed in Section 2. Section 3 illustrates its basic part, a meta model stack. Several use cases of change are analyzed in Section 4; Section 5 finally discusses related work.

2 Foundations

The foundation of Perspective Oriented Process Modelling (POPM) is presented in [10] and [11]; runtime and visualization aspects of POPM are discussed in [12] and [13], respectively. Since POPM combines a couple of matured modelling concepts in a new and synergetic manner, these modelling concepts will be introduced.

2.1 Layered Meta Modelling

In Literature, the term “Meta Model” is often defined as a model of models – e.g. [23]. Thus a meta model defines the structure of models and can be seen as language for defining models. We also use a model to define the structure (syntax) of our process modelling languages within the POPM framework. According to the Meta Object Facility (MOF) [19] this model then becomes part of a meta model stack which consists of several, linearly ordered layers. Since MOF restricts modellers to a specific set of features (e.g. it allows only instanceOf relationships between layers) which is not sufficient for our purpose, our solution is only inspired by them.

In **Fig. 1**, the actual process models are defined on modelling layer M1 (right boxes). A process model uses process (and data, organization etc.) definitions which are collected in the “Type library” on M1 (left box). All process types are defined first and then "used" in process models (e.g. as sub-processes) to define the latter. M0 contains running instances of processes defined on M1 (right boxes).

All process definitions on M1 are defined in a DSL previously specified at M2. M2 further contains the definition of an abstract process meta model (APMM) defining a set of general language features such as Processes, Data Flow or Control Flow. Each DSL is a specialization of elements contained in the APMM. M2 is therefore the layer where a modelling language like BPMN (left boxes) and its derivations (cf. Section 1, right boxes) are defined. It is noteworthy to mention that the elements of M2 reference those on M3 (MOF only allows “instanceOf” relationships).

An abstract process meta meta model (APM^2M) at M3 defines basic modelling principles; for instance, it defines that processes are modelled as directed graphs that also support nesting of nodes; this defines the fundamental structure for process modelling languages. Following the architecture of **Fig. 1** (logical stack) allows for exchanging the modelling paradigm (graph based process models) at M3, defining DSLs at M2 as specializations of a general modelling language (APMM) and establishing type libraries at M1 which allow the re-use of existing process types in

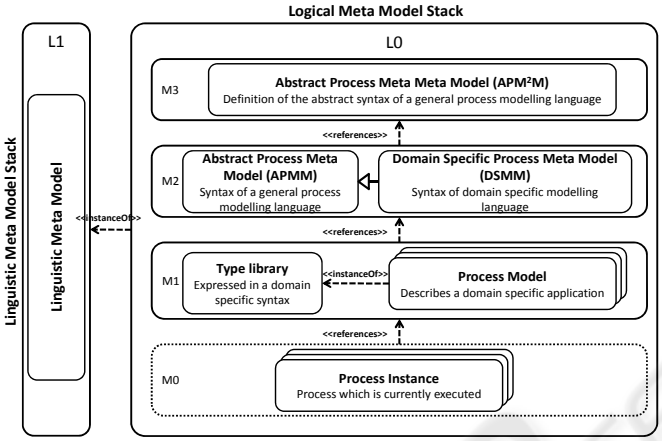

Fig. 1. Meta layer stack of POPM.

different contexts. One of the most powerful features of our approach is that most of the above mentioned kinds of changes can be applied by users and do not require a re-implementation of the modelling tool.

Without going into details we want to introduce one more feature which is most relevant for multi layer modelling. We borrow the Deep Instantiation pattern from [1] that allows defining on which level of a modelling hierarchy a type or an attribute of a type must be instantiated. Thus it is possible to introduce runtime instance identification on M3 that enforces that all derived types must carry this identification. However, this identification is not instantiated before M0.

2.2 Extended Powertypes

As mentioned, the APM^{2}M on M3 defines process models to be interpreted as graphs; for a tool it is then often necessary to interpret the capabilities (features) of each element. For example, both "Process" and "Start-Interface" are nodes of a process model graph. As in a graph usually each node can be connected with others, also the Start-Interface could have incoming arcs; a fact that needs to be prohibited. Therefore already at modelling layer M3 a capability *canHaveIncomingControlFlows* can be defined that describes whether a node accepts incoming flows or not.

Traditional approaches for implementing these capabilities are class hierarchies or constraint languages such as the Object Constraint Language (OCL) [20]. Both approaches are not very useful since either the complexity of the required type hierarchy explodes with an increasing number of capabilities or the user, who should be the one to extend the language, must be familiar with an additional language such as the OCL. Therefore we have chosen to extend the Powertype modelling pattern introduced by Odell in [21]. In our extension the capabilities (e.g. to have incoming flows) are defined as attributes of the powertype. These values then specify which capabilities of the partitioned type should be activated. Furthermore, only those attributes of the partitioned type are inherited by new constructs whose capability

attribute has been set to “true”. Thus our extension removes features physically from a new construct. Complex runtime checks that deal with temporarily disabled features can be omitted this way.

2.3 Logical and Linguistical Modelling

In [2] an orthogonal classification approach is introduced. It contains two stacks that are orthogonal to each other (cf. Fig. 1, Linguistic Meta Model Stack). The Linguistic Meta Model Stack contains a meta model describing how models (including meta models) of the application domain are stored. An orthogonal Logical Meta Model Stack hosts one or more models which are purely content related.

It is crucial for this architecture that each layer of the logical stack can be expressed in the same linguistic model. As a result a modelling tool can be built that allows users to modify all layers of the logical stack in the same way. Conventional modelling tools do not support an explicit linguistic model and thus can usually modify only one layer of a logical model hierarchy [2]. Therefore a profound linguistic meta model is a good basis for creating a modelling tool that allows users to modify arbitrary layers and models.

Due to our extension of the powertype construct, the problem oriented conception of the meta model stack of the logical model, and the application of the orthogonal classification approach, a powerful foundation for an infrastructure for domain specific modelling tools is created. The following sections detail this infrastructure with respect to the most important part – the logical meta model stack.

3 Content of the Logical Meta Model Stack

Our goal is to implement a tool for the POPM framework that is capable of handling changes on the various levels of our meta modelling hierarchy. In this section we will introduce the logical models our actual implementation is based on.

3.1 Abstract Process Meta Meta Model (APM^{2}M)

As explained, the APM^{2}M located at M3 provides basic structures for process modelling languages defined on layer M2, i.e. it prescribes the structure of the modelling elements a process modelling language can offer. The most common graphical notation for process models in POPM is based on directed graphs whose meta meta model is depicted in **Fig. 2** (standard UML notation). It is important to differentiate between modelling and visualization in this context. In **Fig. 2** only the (content related) structure of a process modelling language – and respectively the process models derived from it – is defined. How these models are visualized is not part of this model; visualization is defined in an independent – but certainly related and integrated – model that is published in parts in [13].

Nodes of a process graph are represented by Node in the APM^{2}M (**Fig. 2**). NodeKind then describes the characteristics (features) of nodes in the graph where

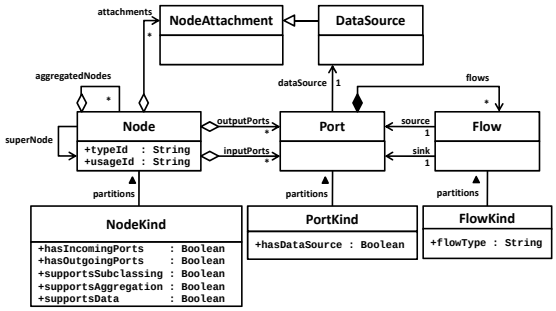

Fig. 2. APM^{2}M of POPM.

each feature corresponds to an attribute of NodeKind. The Powertype pattern between Node and NodeKind is established through the "partitions" relationship; Node represents the partitioned type and NodeKind is the powertype of the Powertype pattern.

Processes are just one type of nodes in such a graph; another type of nodes is e.g. Start-Interface. The different behaviours and capabilities of these two types are determined by the attributes within NodeKind. Features defined and implemented by the partitioned type Node are:

- HasIncomingPorts determines whether a modelling construct can be a destination of incoming flows. It is deactivated for constructs defining the start of a process (Start-Interface).
- HasOutgoingPorts defines if a modelling construct can be the origin of flows. For example a "Stop" interface cannot have outgoing connections.
- SupportsData specifies whether a construct accepts inbound and outbound data flows. If this feature is set to "false" but any of the has…Ports feature attributes has been set to "true", this defines connectivity through control flow(s) only.
- SupportsSubclassing determines if a construct can have another construct as "super type". The child construct will then inherit all attributes from the parent.
- SupportsAggregation defines whether a construct can contain usages of other elements. Typically this feature is activated for process steps but not for interfaces. Thus if activated, hierarchies of modelling elements can be built.

In summary, the features presented above determine whether elements of Node can establish relationships of a certain kind (e.g. superNode, aggregatedNodes, inputPorts) to other types of the APM^{2}M. The extended Powertype concept is also used for the type PortKind – here it determines whether a port can be bound to data sources; FlowKind is using the normal Powertype semantics.

3.2 Abstract Process Meta Model (APMM)

Fig. 3 shows the APMM of POPM, which defines the fundamental components of a POPM-related process model: process, connector, data container, control and data flow, organization, etc.

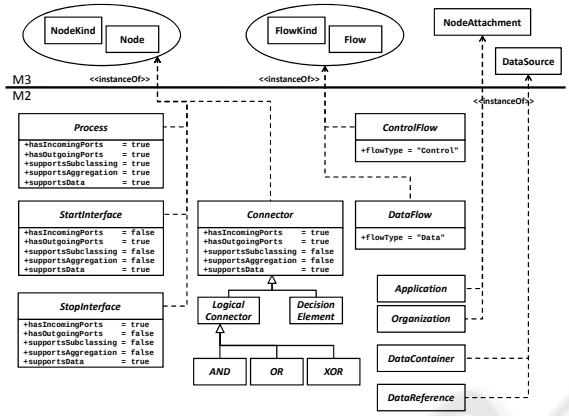

Fig. 3. The core of the Abstract Process Meta Model of POPM.

In the APMM a process is an element in a graph that can be interconnected with other nodes (hasIncoming/OutgoingPorts = true), can receive and produce data (supportsData = true), can be defined in terms of an already existing process (supportsSubclassing = true) and can be used as a container for other elements (supportsAggregation = true). A process – and in general every element on layer M2 – is an instance of a corresponding type (sometimes a powertype) on M3. For instance, Process is an instance of the powertype NodeKind and inherits all activated features from the partitioned type Node. The type StartInterface is also an instance of the powertype NodeKind but does neither support the creation of hierarchies (supportsAggregation = false) nor incoming connections (hasIncomingPorts = false).

3.3 Domain Specific Meta Models (DSMMs)

According to **Fig. 1**, DSMMs are specializations of the APMM. As with object oriented programming languages, abstract types cannot be instantiated. Thus, a DSMM must first provide specializations for each element of the APMM (abstract model) which can be instantiated. Then it can be enriched by additional modelling constructs which determine its specific characteristics. We will show a simple example DSMM from the medical domain in the following.

We decided to provide for each modelling element of the APMM at least one modelling element in the DSMM for the medical domain. These domain specific modelling elements can furthermore be modified in order to capture specific characteristics of the medical realm. For instance the attribute stepType for the modelling element Medical Process (specialization of the APMM element Process) is introduced to determine whether a given step is an administrational or a medical task. Also tags as requested by [16] can be implemented in this way. Completely new modelling constructs can be introduced as well, like the so-called MedicalDecisionElement. In Section 4 we detail this feature.

At level M1 the "normal" modelling of processes takes place. Real (medical) processes use the types defined in the DSMM on M2; for example each process uses

MedicalProcess as basis. Accordingly, input and output data for each process can be defined; the same applies to organizations and operations. In **Fig. 4**c an example is shown. Note that all modelling elements must be defined before being used. For instance, the process Anamnesis must be modelled (and put into the type library) before it can be used as sub-processes within HipTEP.

3.4 Modelling Processes on Level M1

In **Fig. 4**c, a part of a real-world process HipTEP [7] which describes a hip surgery is depicted. It consists of a start interface and two process steps namely Anamnesis and Surgery. The start interface is connected with the Anamnesis step via a control flow whereas Anamnesis and Surgery are also connected with data flows indicating the transport of data items between them. The symbols (document, red cross) inside the two steps are tags that indicate whether a step is more of medical or administrational interest (this is valuable information when the process model has to be analyzed). The tags correspond to the attribute stepType defined in the Medical DSMM for MedicalProcess.

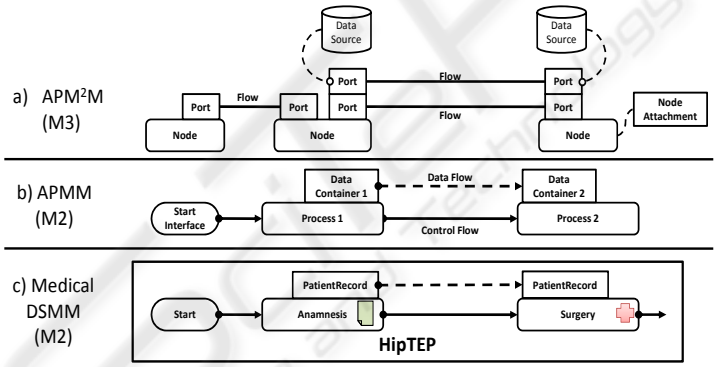

Fig. 4. Stepwise design of a process models on M1.

3.5 Stepwise Design of a Process Model

In **Fig. 4** the three decisive layers of a flexible modelling tool are clearly arranged. The figure illustrates how concepts evolve from very abstract (APM^2M), to more concrete (APMM), to domain specific (DSMM). Some of the metamorphoses of modelling elements are explained in detail.

M3 defines that nodes exist which carry ports (**Fig. 4**a). Ports are sometimes connected with data sources and can be interconnected by Flows. In the derived APMM (**Fig. 4**b) this definition is refined. Nodes are divided into two kinds: StartInterfaces and Processes. Ports which are not connected to data containers have evolved into gluing points for control flows between nodes (StartInterface and Process). Ports connected to data sources demarcate output from input data container for processes which are connected by data flows. **Fig. 4**c then depicts a concrete

example written in the language predetermined by the APMM of **Fig. 4**b. A part of a medical process (HipTEP) is shown which consists of the processes Anamnesis and Surgery. One data item is passed between these processes, namely PatientRecord.

Fig. 4 demonstrates the power of this approach since each artefact of a process model is explicitly defined on clearly separated meta levels.

4 Dealing with Change

We will now explain concrete use cases of changes. These scenarios are ordered according to their relevance in practice based on our experience. We also depict how users can use them in a safe and structured way.

4.1 Change I: New Feature for an Existing Construct (Tagging)

Often it is necessary to distinguish processes from each other. Frequently, special tags are attached to processes and visualized in a suitable form [7] [16]. Speaking in terms of our logical meta model stack this means that an attribute is added to the corresponding modelling element in the DSMM that holds the tag. In Section 3 we have already shown this extension by adding the stepType attribute to the MedicalProcess type. Depending on the actual value of this attribute a visualization algorithm can then e.g. display icons appropriately.

4.2 Change II: Introducing New Constructs

One reason for adapting modelling constructs is the evolution of the application domain. For example, due to more insight into the domain more powerful and semantically richer modelling constructs have to be created.

A new construct can either be defined “from scratch” or by redefining an already existing constructs of the DSMM or APMM. Fig. 5 gives an example for this kind of change in the medical domain. **Fig. 5**a outlines the complex structure of a medical decision path whereas **Fig. 5**b depicts a newly created modelling construct *MedicalDecisionElement* which is a macro comprising the functionality of the complex process structure of **Fig. 5**a. The problem with the process in **Fig. 5**a is that it is not comprehensible easily (only the complex structure of the decision path is of interest; therefore we did not show any details in **Fig. 5**a). Thus we decided to introduce a new compact modelling construct MedicalDecisionElement (**Fig. 5**b). This construct comprises the same functionality but is much easier to interpret. First, the construct has a title clearly showing its purpose. Then the most interesting decisions are shown in the list below the title and the two possible outcomes – yes or no – are depicted on the right side. The introduction of this compact construct – together with the consequent elimination of unreadable process models – was one of the major factors why process modelling was accepted as adequate means to illustrate the medical applications in the Ophthalmological Clinics of the University of Erlangen [14]. This project convincingly demonstrated that a domain specific

modelling language is not just "nice-to-have" but is crucial for the acceptance of process management in general.

4.3 Change III: Enhancing / Changing the Modelling Method

So far all changes of process modelling languages were applied to DSLs individually.

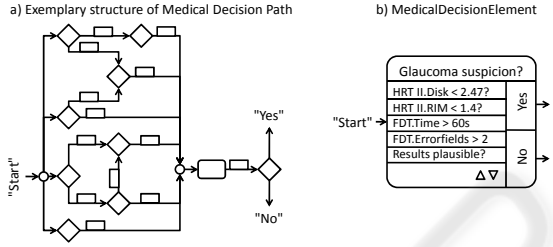

Fig. 5. The MedicalDecisionElement (b) subsumes many single decisions (a).

In our approach it is also possible to change the modelling method as such. This change happens on layer M3 and affects all process modelling languages defined below. For instance, from now on we will prohibit control flows between nodes. Referring to the APM^{2}M in **Fig. 4** this means to remove ports which are not connected with data sources. Consequently all flow derived from this constellation must be removed from all process modelling languages on M2 and also from all defined process models on M1.

5 Related Work

We now give an overview on existing technologies and systems (beside those already introduced in Section 2) that aim at increasing the sustainability of information systems. We will show that these are – per se – not appropriate for domain experts because they require extensive programming skills or are not flexible enough.

Generative Programming [5] and Software Factories [8] are techniques for the reuse of code. Generative Programming aims at the generation of code out of a set of templates. Requiring programming skills to produce valid and correct results, Generative Programming is unusable for end-users or domain experts. Software Factories in contrast aim at reducing the cost factors (time, resources etc.) during application development. This again is not suitable for end-users or domain experts. Even more harmful is that both approaches are meant to be applied during the development phase of an application but not during runtime.

Beside programming techniques, we also investigated complete meta modelling systems e.g. the Microsoft Domain Specific Language Tools for Visual Studio [17], the Eclipse Modeling Framework (EMF) [6] (along with related technologies that support the generation of graphical editors) or MetaEdit+ [15]. Most of them use only two levels in which the type level defines the storage format for the user models.

Beside this the modelling freedom is restricted by a fixed underlying meta model. Also many solutions are not able to use a new modelling language without generating a new modelling environment.

Summarizing, there are solutions that provide some means for building modelling tools. But either they require too much programming skills or they are not flexible enough.

6 Conclusions

In this paper we introduced our approach for a more sustainable process modelling environment that can be easily adapted by domain experts to their realm without programming in general. We showed that many concepts exist which can already be used to establish flexible and adaptable systems but which unfold their real power after they were combined to form one unified and comprehensive approach. We have then shown how different adaptation scenarios can be performed with the help of these concepts. Here the important key-point is that all those change requests that are most common can be performed without writing code; instead only a new configuration for the system has to be provided which is easy to set up even though the domain expert who is pursuing these changes has not much knowledge about the system internals. Thus domain experts are empowered to adapt the whole system perpetually to changing requirements which we believe is a fundamental step towards more sustainability.

References

1. Atkinson, C., Kühne, T.: The Essence of Multilevel Metamodeling. Proc. of the 4th Int'l Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools. Springer-Verlag, Toronto, Canada (2001)
2. Atkinson, C., Kühne, T.: Concepts for Comparing Modeling Tool Architectures. Lecture Notes in Computer Science. (2005) 398-413
3. Clarence, E., Karim, K., Grzegorz, R.: Dynamic change within workflow systems. Conference on Organizational Computing Systems. ACM, Milpitas, California, United States (1995)
4. Clark, T., Sammut, P., Willans, J.: Applied Metamodelling - A Foundation For Language Driven Development. CETEVA (2008)
5. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley Professional (2000)
6. Eclipse Foundation: Eclipse Modeling Framework Project (EMF). http://www.eclipse.org/modeling/emf/?project=emf [2008-11-26]
7. Faerber, M., Jablonski, S., Schneider, T.: A Comprehensive Modeling Language for Clinical Processes. Proc. of the European Conference on eHealth 2007, Lecture Notes in Informatics (LNI), GI, Oldenburg, Germany (2007) 77-88
8. Greenfield, J., Short, K., Cook, S., Kent, S.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Wiley (2004)

9. Heinl, P., Horn, S., Jablonski, S., Neeb, J., Stein, K., Teschke, M.: A comprehensive approach to flexibility in workflow management systems. SIGSOFT Softw. Eng. Notes 24 (1999) 79-88
10. Jablonski, S.: MOBILE: A Modular Workflow Model and Architecture. 4th Int'l Working Conference on Dynamic Modelling and Information Systems Noordwijkerhout, NL (1994)
11. Jablonski, S., Bussler, C.: Workflow Management: Modeling Concepts, Architecture and Implementation. Int'l Thomson Computer Press (1996)
12. Jablonski, S., Faerber, M., Götz, M., Volz, B., Dornstauder, S., Müller, S.: Integrated Process Execution: A Generic Execution Infrastructure for Process Models. 4th Int'l Conference on Business Process Management (BPM), Vienna, Austria (2006)
13. Jablonski, S., Götz, M.: Perspective Oriented Business Process Visualization. Business Process Management Workshops. Springer (2008) 144-155
14. Jablonski, S., Lay, R., Meiler, C., Müller, S., Hümmer, W.: Data logistics as a means of integration in healthcare applications. 2005 ACM symposium on Applied computing. ACM, Santa Fe, New Mexico (2005)
15. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. 8th Int'l Conference on Advanced Information System Engineering. Springer, Heraklion, Crete, Greece (1996) 1-21
16. Lu, R., Sadiq, S.: On the discovery of preferred work practice through business process variants. 26th Int'l Conference on Conceptual Modeling. Springer, Auckland, New Zealand (2007) 165-180
17. Microsoft: Domain-Specific Language Tools. http://msdn.microsoft.com/en-us/library/bb126235.aspx [2008-11-26]
18. Object Management Group: BPMN 1.1 Specification. http://www.omg.org/spec/BPMN/1.1/ [2008-11-26]
19. Object Management Group: MOF 2.0 Specification. http://www.omg.org/spec/MOF/2.0/ [2008-11-26]
20. Object Management Group: OCL 2.0 Specification. http://www.omg.org/spec/OCL/2.0/ [2008-11-26]
21. Odell, J.: Advanced Object-Oriented Analysis and Design Using UML. Cambridge University Press, New York, NY, USA (1998)
22. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems--a survey. Data & Knowledge Engineering 50 (2004) 9-34
23. Seidewitz, E.: What Models Mean. IEEE Software 20 (2003) 26-32
24. van der Aalst, W.M.P., Jablonski, S.: Dealing with workflow change: identification of issues and solutions. International Journal of Computer Systems Science and Engineering 15 (2000) 267-276