Evolution of Business Process Models and Languages

Stefan Jablonski, Bernhard Volz, Sebastian Dornstauder

Chair for Applied Computer Science IV
University of Bayreuth
Universitätsstraße 30
95447 Bayreuth, Germany
{stefan.jablonski,bernhard.volz,sebastian.dornstauder}@uni-bayreuth.de

Abstract: "The only constant is change" is an often cited phrase. We regard it as predominant for the area of process based information systems. In this paper we investigate how evolution of process based information systems can be supported by a process modeling framework that easily can be adjusted to changing requirements of an application domain. Our key contribution is the provision of a system infrastructure that supports the adaptation of both process modeling languages and process models to evolving application requirements. Our approach is based on a multi level meta modeling framework.

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 modeling hierarchy (Section 3), the phenomena of change can be classified according to the process modeling level it occurs. Starting at the "lowest" level, running process instances might have to be changed to react to a sudden shift in the application. Among others, [Aa00], [El95] and [Ri04] are investigating this issue and suggest adequate solutions. Stepping one level up, the process type (process model, process 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 [Aa00] [He99]. Nevertheless, it is possible to even step up another level in a process modeling hierarchy. Change on this level means altering the modeling language used to define process models. We focus this third interpretation of change in this paper and investigate changes of process modeling languages (PMLs) since this is the kind of change which is not sufficiently dealt with in research and promises powerful means to implement process oriented information systems adequately. Nevertheless, we also discuss changes of process models since this is close to the change of a PML. Changing process instances is neglected because it is already discussed broadly in [Ja06].

Why is the change of a PML an issue that is worth to be investigated? One can argue that a PML should always remain untouched. However, we fully comply with the interpretation of change as given in [Cl08]; there, change is related to diversity. The authors of [Cl08] state that "life would be much easier if there was only one programming language and one deployment platform". They 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 represented by the modeling language and the deployment platform corresponds to the process execution infrastructure.

We fully subscribe to the argument of [Cl08] that the right (process modeling) 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). 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 domain specific languages – a kind of comparability and compatibility between them. We finally agree that meta modeling provides capabilities to achieve this.

We started with the discussion of change and ended up in domain specific process modeling languages (DSPMLs). This is due to the fact that a DSPML is evolving from a standard PML. Thus, from a technical perspective evolution means changing models over time. Due to our specialized architecture being able to cope with evolution of a PML means to be also capable to cope with the evolution of process models.

We present a meta modeling approach which supports the definition of DSPMLs. The special feature of our approach is that domain specific languages 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 shows major advantages.

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

The focus of this paper is on developing tools that support evolution of PMLs and process models. As we have discussed before, domain specific languages play an important role in that scenario. The foundation of a domain specific processes modeling tool is discussed in Section 2. Section 3 then illustrates its basic part, a meta model stack. Several use cases of evolution are analyzed in Section 4. Section 5 finally discusses related work.

2 The Foundation of Perspective Oriented Process Modeling

Perspective Oriented Process Modeling (POPM) is presented in general in [Ja94] and [JB96]. The runtime and visualization aspects of POPM are discussed in more detail in [Ja06] and [JG07], respectively. Since POPM itself is subject of an ongoing evolution, POPM now combines a couple of matured modeling concepts in a new and synergetic manner. These modeling concepts are introduced in the following.

2.1 Layered Meta Modeling

Meta modeling techniques are commonly used to describe the structure of models; thus "Meta Model" is often defined as "model of a modeling language" (e.g. [Se03]). The system under study in our case is a model, for instance a process model "Travel Claim Reimbursement Process". A PML has to be used to describe this process, i.e. this process model. We also use a model to describe such a PML; this model is indeed a meta model defining the structure (syntax) of our PMLs within the POPM framework. According to the Meta Object Facility (MOF, [OMG06a]) this model then becomes part of a meta model stack which consists of several, linearly ordered layers. Since MOF restricts modelers to a specific set of features which is not sufficient for our purpose, our solution is not strictly following the modeling rules of MOF.

In Fig. 1 (meta model stack of our POPM framework), actual process models are defined on M1 (right boxes). A process model uses process definitions (and data definitions, organization definitions etc.) which are all gathered in the "Type library" on M1 (left box). It is noteworthy to mention that on M1 the concept "type/usage" is applied: process types are defined (and put into the type library) and then are "used" in other process models (e.g. as sub-processes) to define the latter. M0 contains running instances of process types formerly defined on M1 (right boxes).

All process definitions on M1 are defined in an application specific language which must have previously been defined at M2. M2 contains the definition of an abstract process meta model (APMM) that defines a set of general language features, e.g. a standard language like BPMN. Basic concepts such as Processes, Data Flow or Control Flow are defined in such an APMM. All domain specific language dialects can also be found on M2 as a specialization of the APMM.

An abstract process meta meta model (APM^2M) at M3 defines basic modeling principles; for instance, it is defined that processes consist out of "bubbles and arcs" (directed graph) or that nesting of modeling elements is allowed. This APM^2M thus defines the fundamental structure, i.e. the structural templates, for PMLs specified on the lower level.

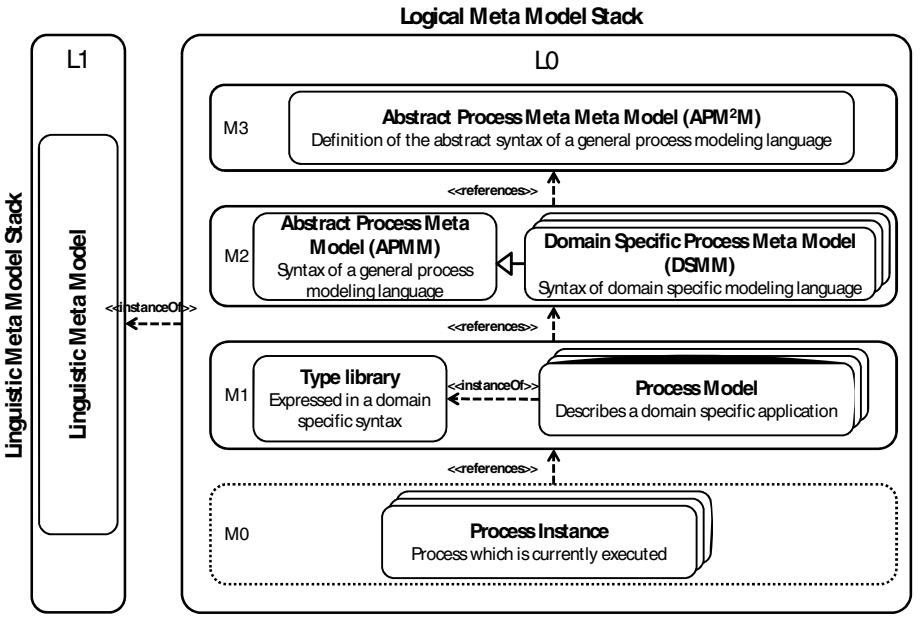

Fig. 1. The meta model stack of POPM

Following the architecture of Fig. 1 (Logical Meta Model Stack) allows for

- exchanging the modeling paradigm (graph based process models) at M3,
- defining DSPMLs at M2 as specializations of a general modeling language (APMM) and
- adapting process models at M1.

We believe that these capabilities provide a powerful basis for the evolution of PMLs and process models.

2.2 Extended Powertypes

We have mentioned that the APM^{2}M on M3 defines process models to be interpreted as graphs; for a tool it is then often necessary not only to recognize each element of such a graph together with its attributes (and operations) but also to know the capabilities (features) of each element. For example, both "Process" and "Start-Interface" are nodes of a process model graph. As in a graph each node can be connected with other nodes in principle, also the Start-Interface could have an incoming Control Flow arc. Obviously this needs to be prohibited since the Start-Interface denotes the beginning of a process and thus should not receive any incoming flows at all. Thus a capability "canHaveIncomingControlFlows" can be defined at M3. Such a capability (e.g. to have incoming flows) is defined as an attribute of the powertype. These values define which capabilities of the second type of the pattern, the so-called partitioned type, should be activated (they must be set to "true"). Furthermore only those attributes of the partitioned type are inherited by new constructs whose capability attribute has been set to "true". Thus our extension does not only activate or deactivate a set of certain features but also removes them physically from new constructs such that complex runtime checks dealing with disabled features can be avoided. But the main benefit is that this pattern eases the definition of completely new modeling constructs at M2 since the user can define which features are supported by a new construct easily.

2.3 Logical and Linguistic Modeling

In [AK05] an orthogonal classification approach is introduced; this approach contains two meta stacks instead of only one which are orthogonal to each other (cf. Fig. 1, Linguistic Meta Model Stack). One stack contains a meta model that describes how models (of the application domain that have to be defined) can be stored (linguistic model, e.g. "how is an attribute stored"). The other stack hosts the logical model which is 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 modeling tool can be built that allows users to modify all layers of the logical stack in the same way since all logical layers are described by the same linguistic model. This is not the usual way modeling tools are built. Conventional modeling tools do not support an explicit linguistic model and thus can usually modify only one layer of a logical model hierarchy [AK05]. Therefore more than one tool is required if both, PMLs and process models should be modified. As the number of required modeling tools is directly proportional to the number of domain specific languages, it is better to provide only modeling tool that supports many DSLs along with the ability to modify process models defined in the corresponding DSLs.

Our goal is to implement a tool for the POPM framework that is capable of handling evolution on the various levels of our meta modeling hierarchy. In this section we will introduce the basic concepts for this tool implementation. Section 4 demonstrates how evolution can be accommodated by the tool.

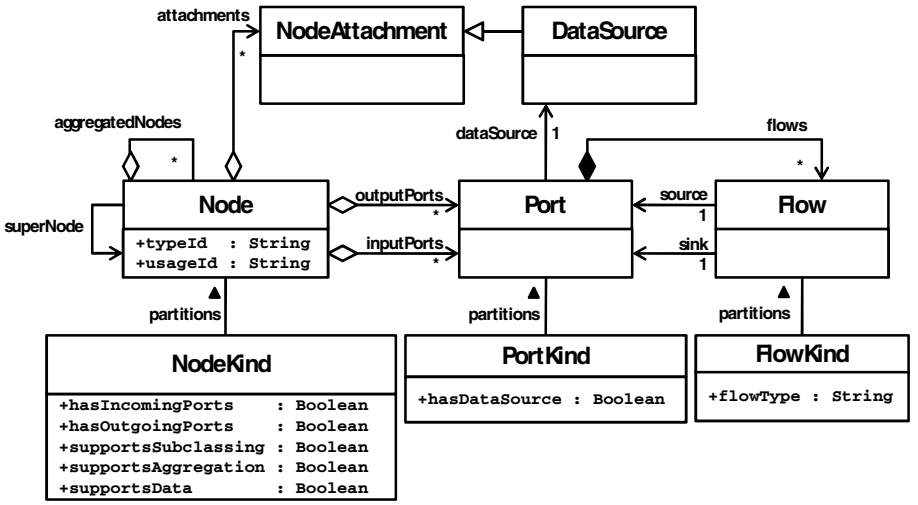

Fig. 2. The APM^{2}M – defining the fundamental modeling method

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

As explained in Section 2, the APM^{2}M is located at M3 and provides basic structures for PMLs defined on M2, i.e. it prescribes the structure of all modeling elements of a PML. The most common graphical notation for process models in POPM is the "bubbles & arcs" notation whose meta meta model is depicted in Fig. 2 (standard UML notation). It is important to differentiate between modeling and visualization in this context. In Fig. 2 only the (content related) structure of a PML – and respectively the process models derived from it – is defined. Visualization is defined in an independent – but certainly related and integrated – model (cf. [JG07]).

A process model in POPM can be regarded as graph whereby processes are the nodes of the graph. These nodes are represented by *Node* in the APM^{2}M (Fig. 2). *NodeKind* describes the characteristics (features) of a node where each characteristic corresponds to one attribute of *NodeKind*. The Powertype pattern between *Node* and *NodeKind* is then established with the “partitions” relationship; *Node* represents the partitioned type and *NodeKind* is the powertype of the Powertype pattern. Features can be defined (*NodeKind*) that individually determine the behavior of *Node*. The following featurs are available: *hasIncomingPorts, hasOutgoingPorts, supportsData, supportsSubclassing, supportsAggregation* – their meaning and purpose can easily be derived from their names. In summary, the features presented above determine whether elements of *Node* can establish relationships of a certain kind (e.g. *aggregatedNodes*, *superNode*, *inputPorts*) to other types of the APM^{2}M.

2.5 Abstract Process Meta Model (APMM)

Fig. 3 shows the APMM of POPM; in this model the fundamental components of a POPM-related process model are defined: process, connector, data container, control and data flow, organization etc.

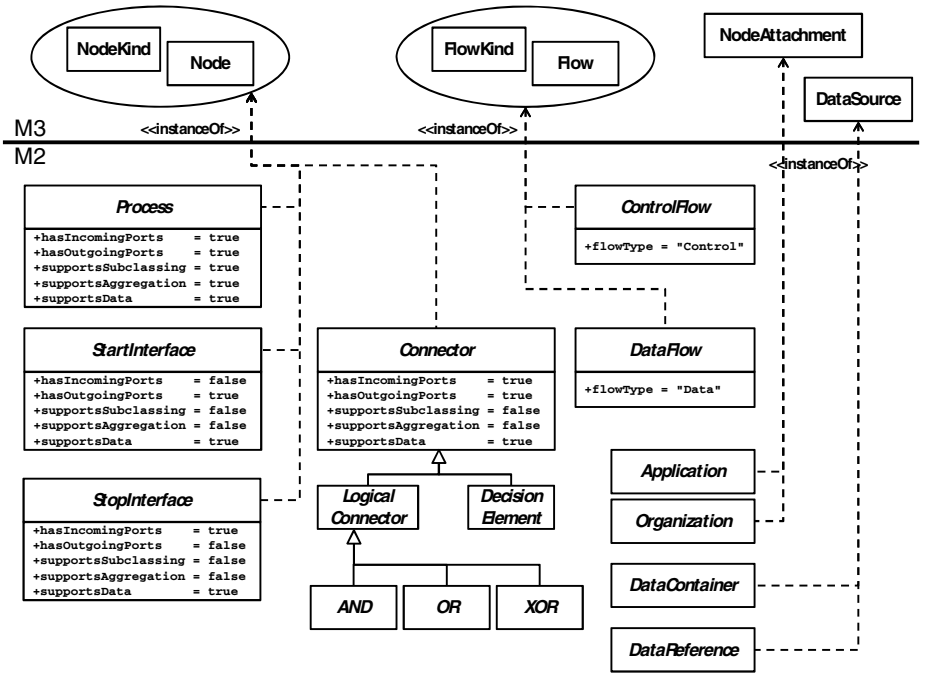

Fig. 3. The core of the Abstract Process Meta Model of POPM

In the APMM a process is an element in a graph that can be interconnected with other nodes (*hasIncomingPorts* = *true*, *hasOutgoingPorts* = *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 of M2 – is an instance of a corresponding type – sometimes this is 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 types *StartInterface* and *StopInterface*) are also instances of the powertype *NodeKind* but do not support the creation of hierarchies since the corresponding feature attribute is not set (*supportsAggregation* = *false*); additionally, a *StartInterface* does not support incoming connections (*hasIncomingPorts* = *false*); analogously a *StopInterface* does not allow outgoing connections (*hasOutgoingPorts* = *false*).

2.6 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, a DSMM can be enriched by additional modeling constructs which determine its specific characteristics. Fig. 4 shows an example DSMM from the medical realm. This figure also depicts how domain specific modeling elements can furthermore be modified in order to capture specific characteristics. The attribute *stepType* for the modeling element *Medical Process* is introduced to determine whether a given step is an administrational task or a medical task; according to [Fa07] this is a fundamental distinction. Also the tags requested in [LS07] can be implemented in this way.

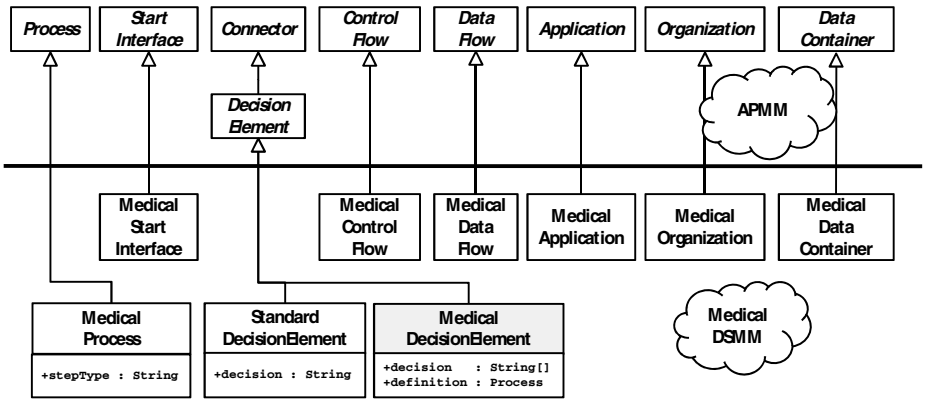

Fig. 4. A DSMM from the medical realm

It is possible to introduce completely new modeling constructs on this level as well. In Fig. 4, the so-called *MedicalDecisionElement* is presented. This modeling construct represents a complex series of single decisions.

2.7 Modeling Processes on M1

At level M1 "normal" process modeling takes place. We assume that a DSMM is defined on M2. Then real processes can be modeled on M1 and which are all derived from *MedicalProcess*. Accordingly, input and output data for each process can be defined; the same applies to organizations and operations. In Fig. 5c an example is shown. Regard, all modeling elements must be defined first, before they can be used within a process model. The process model consists of a start interface and two process steps namely *Anamnesis* and *Surgery*. 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*.

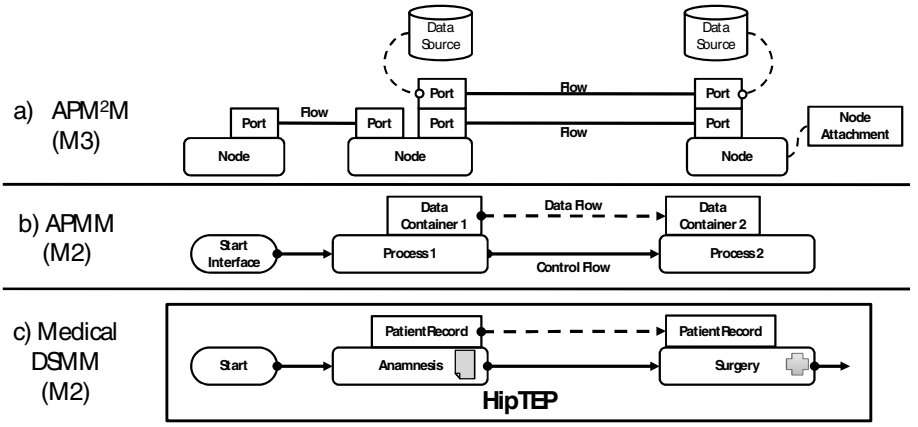

Fig. 5. Stepwise design of a process model

2.8 Stepwise design of a process model

In Fig. 5 the three decisive layers of a flexible modeling tool are clearly arranged. The figure illustrates nicely how concepts are evolving from very abstract (APM^2M), to more concrete (APMM), to domain specific (DSMM). Some of the metamorphoses of modeling elements are explained in detail. Through our integrated approach – models on M3, M2 and M1 can be manipulated by the same tool – consistency is best guaranteed on all layers of the meta modeling stack.

3 Evolution of process modeling languages and process models

We will now explain seven concrete use cases for evolution. These scenarios are sorted according to their relevance and frequency of occurrence in practice based on our experience gained in many industrial projects. The changes range over all levels of our logical meta stack; Change I affects M1, Changes II, III and IV concern M2 and Change V works on M3. A special role play Changes VI and VII; both affect layers not depicted in our meta hierarchy. We included them for completeness, however.

3.1 Change I - Adapting an existing process model

Adapting a process model to changing application requirements is a very frequent task in practice. Typical examples are the exchange of an application inside a process, the change of the execution order or the introduction / deletion of work steps. This kind of change is the normal use case for a modeling tool. It just has to be decided whether the generated new process models are replacements, versions or variants of the original process model. For process modelers this kind of change is uncritical.

3.2 Change II – Introducing new features for process modeling constructs (tagging)

Often it is necessary to distinguish processes from each other. Therefore, special tags are attached to processes and visualized in a suitable form [Fa07] [LS07]. Speaking in terms of our logical meta model stack this means that an attribute is added to the corresponding modeling element in the DSMM for storing the tag. In Section 3 we have shown such an extension: there the attribute *stepType* was added to the *MedicalProcess* type which was not part of the standard modeling. Depending on the actual value of this attribute a visualization algorithm can then for example display icons. The enactment of such a change can easily be performed by a domain expert.

3.3 Change III – Introducing a new process modeling construct

Modeling constructs have to be changed due an evolution of the application domain. For example, more powerful and semantically richer modeling constructs have to be created. Process modeling constructs are located at M2, usually in a specific DSMM. A new construct can either be defined "from scratch" or by redefining an already existing construct of the DSMM or APMM.

In case the new construct is defined from scratch, the creation task is facilitated enormously by the extended powertype concept. Merely a new construct must be defined on M2. If it is a kind of a process node it can inherit functionality provided by *Node; to selectively inherit this functionality* proper values for the feature attributes of the powertype *NodeKind* must be set. Nevertheless, this modification of a modeling language can be performed by any domain expert.

A new modeling construct can also be based on existing constructs like the medical decider from the medical application domain (Section 3). It is based on the standard decision construct of the APMM and summarizes a variety of single decisions into one compound construct (Fig. 6).

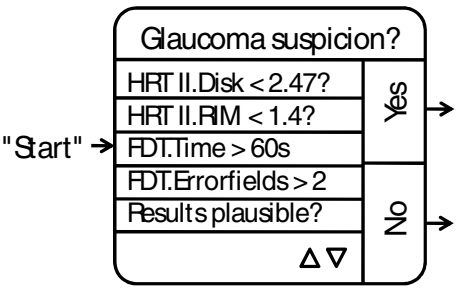

Fig. 6. The Medical Decision Element

In contrast to the standard decision construct the Medical Decision Element comprises many single decisions. Further, the output of the medical decision element is either "Yes" or "No" instead of an arbitrary result of the standard decision element. The introduction of this compact construct was one of the major factors why process modeling was accepted as adequate means to illustrate the medical applications in the Ophthalmological Clinics of the University of Erlangen [Ja05] where complex medical decisions had to be modeled often. This project convincingly demonstrated that a domain specific modeling language is not just "nice-to-have" but is crucial for the acceptance of process management in general. This kind of change corresponds to the introduction of macros in modeling languages.

3.4 Change IV – Adding a new perspective

Perspectives are in general defined at M2 – in the APMM for all domains as well as in DSPMMs for a single one. Similar to modeling constructs, new perspectives can be introduced in both. For example, it might become necessary to argue about process execution times in detail. So, various kinds of time should be introduced: preparation time, actual execution time, queue time and post processing time. A new perspective will be defined as an instance of the type *NodeAttachment* which is part of the APM^2M at M3. A perspective comprises multiple new modeling constructs. The constructs of this perspective can be associated with already existing modeling constructs like a process step or a flow construct.

3.5 Change V – Enhancing / changing the modeling method

So far all changes of PMLs were applied to DSLs individually. In our approach it is also possible to change the modeling method as such. This change happens on M3 and affects all PMLs defined below. For instance, from now on we will prohibit control flows between nodes. Referring to the APM^2M in Fig. 5 this means to remove ports which are not connected with data sources. Consequently all flow derived from this constellation must be removed from all PMLs on M2 and also from all defined process models on M1.

Another use case for changing the modeling method is if the used paradigm of directed cyclic graphs should be exchanged to something else, for instance petri nets. Also petri nets are graphs but they do not have the distinction of nodes, node attachments and flows. Instead they use states and transitions. Thus the presented model on M3 will not fit anymore and must be exchanged against a meta model for petri nets.

3.6 Change VI – Applying new semantics for a modeling construct

Without going into detail this change should be introduced. The idea is to change execution semantics for a process construct. For example, instead of interpreting a control flow arrow between two process steps as a mandatory order it should from now on be interpreted as recommended order. Details of the implementation of such kind of evolution can be found in [JI08] and [Ja06].

3.7 Change VII – Adapting the graphical representation of a construct

We were already mentioning that all models so far are just presenting conceptual issues. Visualization of process modeling constructs was excluded until now. In [JG07] we show how different kinds of visualizations can be associated with process modeling constructs according to evolving application requirements.

4 Related Work

We now give an overview on existing technologies and systems (beside those already introduced in Section 2) that aim at increasing the adaptability 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 [Cz00] and Software Factories [Gr99] are techniques for the reuse of code as known from object-oriented or component-oriented programming. Generative Programming aims at the generation of code out of a generic set of templates. These templates along with a specification of the outcome are handed to a code generator that automatically produces code. Because of the programming skills required to produce valid and correct results, Generative Programming is unusable for end-users or domain experts. Software Factories in contrast aim at reducing the costs (time, resources etc.) during the development of an application. This it is again an approach which is suitable for software developers but not 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 such that it is possible to introduce changes but a re-compilation and a re-deployment is needed.

Beside these programming techniques also complete systems exist that empower the user to build models such as the Microsoft Domain Specific Language Tools for Visual Studio [Mi07], the Eclipse Modeling Framework (EMF) [Ec07] (with related technologies for the generation of graphical editors) or MetaEdit+ [Ke96]. But nearly all of them use only two layers (definition of user model and instances of these) in which the type level defines the storage format for the user models. Beside this the modeling freedom is restricted as the underlying meta models often is fixed. For example EMF uses a subset of MOF and thus also inherits many restrictions from it – one is that it is not possible to use powertypes. Also many solutions are not able to use a new modeling language without generating a new modeling environment explicitly. Furthermore concepts for supporting more than one view are less sophisticated – if existing at all. But there are also some commonalities with these approaches from a technical point of view. We can reuse frameworks for code generation and graphical modeling that have already proven their strengths for meta modeling applications.

5 Conclusion

In this paper we were introducing our approach of a powerful process modeling infrastructure that supports evolution of PMLs and process models. We showed that we can leverage on some interesting concepts which unfold their real power after they were combined to form this unified and comprehensive approach. We have then shown how different evolution 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. Thus domain experts are empowered to develop their modeling language(s) and applications by themselves and thus keep adapting the whole system perpetually to changing requirements. So as the overall system is able to handle changes by itself and it can be permanently improved by domain experts, we believe that this method supports sustainability greatly.

6 References

[Aa00] v.d. Aalst, W.; Jablonski, S.: Dealing with workflow change: identification of issues and solutions. International Journal of Computer Systems Science & Engineering (CSSE), Vol. 15 (2000), No. 5, 267 - 276

[AK01] Atkinson, C.; Kühne, T.: The Essence of Multilevel Metamodeling, 4th Int'l Conference on the Unified Modeling Language, Toronto, Canada, 1-5, 10.2001

[AK05] Atkinson, C.; Kühne, T.: Concepts for Comparing Modeling Tool Architectures, ACM/IEEE 8th Int'l Conference on Model Driven Engineering Languages and Systems, MoDELS / UML 2005, October 2-7, Montego Bay, Jamaica, 2005

[BS08] Berghoff, W.M.; Schlichter, J.H.: Computer-Supported Cooperative Work: Introduction to Distributed Applications. Springer-Verlag, 2000

[Cl08] Clark, T.; Sammut, P.; Willians, J.: Applied Metamodelling – A Foundation For Language Driven Development, 2nd Edition, CETEVA 2008, (visited: 2008-03-12) http://www.ceteva.com/book.html

[Cz00] Czarnecki, K.; Eisenecker, U.: Generative Programming: method, tools and applications. Addison-Wesley, 2000

[Ec07] The Eclipse Foundation: Eclipse Modeling Framework, (visited: 2007-03-29) http://www.eclipse.org/modeling/emf/?project=emf

[El95] Ellis, C., K. Keddara and G. Rozenberg (1995): Dynamic Change within Workflow Systems. In N. Comstock et al. (eds.): Proceedings of Conference on Organizational Computing Systems (COOCS'95). New York: ACM, pp. 10–21.

[Fa07] Faerber, M.; Jablonski, S.; Schneider, T.: A Comprehensive Modeling Language for Clinical Processes. 2nd European Conference on eHealth (ECEH'07), Oldenburg, Germany, 10.2007

[Gr99] Greenfield, J.; Short, K.: Software Factories: assembling applications with patterns, models, frameworks and tools. Wiley Publishing, 2004

[He99] 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(2):79-88, 1999

[Ja94] Jablonski, S.: MOBILE: A Modular Workflow Model and Architecture. Proc. International Working Conference on Dynamic Modeling and Information Systems, Noordwijkerhout, NL, 1994

[JB96] Jablonski, S.; Bussler, C.: Workflow Management – Modeling Concepts, Architecture and Implementation. London: Int. Thomson Computer Press, 1996

[Ja05] Jablonski, S.; Lay, R.; Meiler, C.; Müller, S.; Hümmer, W.: Data Logistics as a Means of Integration in Healthcare Applications. Proc. 2005 ACM Symposium on Applied Computing (SAC) - Special Track on Computer Applications in Health Care, Santa Fe, New Mexico, 03.2005

[JG07] Jablonski, S.; Götz, M.: Perspective Oriented Business Process Visualization. 3rd International Workshop on Business Process Design (BPD) 5th International Conference on Business Process Management (BPM 2007). Brisbane, 9.2007

[JI08] Jablonski, S.; Igler, M.: Flexible Process Modeling and Execution based on Declarative Programming. Technical Report, University of Bayreuth, 2008

[Ja06] Jablonski, S.; Müller, S.; Faerber, M; Götz, M.; Volz, B.; Dornstauder, S.: Integrated Process Execution: A Generic Execution Infrastructure for Process Models. BPM Demo Session, 4th Int'l Confer¬¬¬ence on Business Process Management (BPM 2006). Austria, Vienna, 9.2006.

[Ke96] Kelly, S.; Lyytinen, K.; Rossi, M.: MetaEdit+: A fully configurable multi-user and multi-tool CASE and CAME environment. Proceedings of the 8th International Conference CAISE'96, Springer-Verlag, 1996, pp. 1-21

[LS07] Lu, R.; Sadiq, S.: On the Discovery of Preferred Work Practice Through Business Process Variants, 26th Int'l Conference on Conceptual Modeling (ER 2007), Auckland, New Zealand, 11.2007

[Me04] Melnik, S.: Generic Model Management: concepts and algorithms. Springer, 2004

[Mi07] Microsoft: Domain-Specific Language Tools, (visited: 2007-03-29) http://msdn2.microsoft.com/en-us/vstudio/aa718368.aspx

[OMG06a] Object Management Group: Meta Object Facility Core Specification version 2.0, 2006-01-01

[OMG06b] Object Management Group: Object Constraint Language Specification Version 2.0, 06-05-01

[OMG06c] Object Management Group: Business Process Modeling Notation Specification, 2006-02-01

[Od98] Odell, J.: Advanced Object-Oriented Analysis and Design using UML. Cambridge University Press, 1998

[Pe05] Petrov, I.: Meta-data, Meta-Modelling and Query Processing in Meta-data Repository Systems. PhD thesis, Univ. of Erlangen-Nürnberg, Germany, 12.2005

[Ri04] Rinderle, S.; Reichert, M.; Dadam, P.: Correctness Criteria for Dynamic Changes in Workflow Systems - A Survey. Data and Knowledge Engineering, Special Issue on Advances in Business Process Management 50(1):9-34 (2004)

[Se03] Seidewitz, E.: What models mean. IEEE Software 2003, 20(5): pp. 26-31