# Do We Really Know How to Support Processes? Considerations and Reconstruction

Stefan Jablonski

University of Bayreuth
Institute for Informatics
Stefan.Jablonski@uni-bayreuth.de

**Abstract.** Since approximately 20 years process management is regarded as innovative technology both for the description of complex applications and for supporting their execution. Soon after the beginning of this process era, an inflation of process management systems started to flood the market. However, many users were very frustrated when they had to experience that these process management systems did not support their applications properly. One of the main causes for failing process applications was that process management was totally underestimated. In this contribution we try to get to the bottom of process management, i.e. we reconstruct the real requirements of process management which are the basis for the construction of working process solutions.

**Keywords:** Process Management, Process Modeling, Process Execution, Workflow Management.

## 1 Introduction

Since approximately 20 years, process management is regarded as innovative technology both for the description of complex applications and for supporting their execution. Soon after the beginning of this process era, an inflation of process management systems started to flood the market. However, many users were very frustrated when they had to experience that these process management systems did not support their applications properly.

One of the main causes for failing process applications was that process management was totally underestimated. Defining processes is more than painting "bubbles-and-arcs" pictures. Developing processes requires both a profound methodology and a powerful and expressive process modeling language. Executing processes is more than executing steps in a fixed and predefined order like jobs in a batch queue. Supporting process execution requires an infrastructure that definitely reflects mandatory restrictions imposed by applications but also sufficient flexibility must be provided such that users are not limited in their creativity.

The contribution of this paper is not a new method for process management; rather it intends to reveal fundamental issues of process management (process modeling and process execution). This investigation should inspire researchers to contemplate current developments and research areas; it should foster to develop new concepts for

G. Engels et al. (Eds.): Nagl Festschrift, LNCS 5765, pp. 393–410, 2010.

process management that are better coping with the requirements of process based applications. Especially, it will stimulate researchers to more thoroughly analyze requirements of process based applications. It will therefore not focus on the introduction of a new technical framework to deal with process management; rather, it wants to illustrate unconventional ways of process modeling and execution which foster flexible process management.

Section 2 identifies problems with process modeling and execution. Section 3 then introduces possible solutions to those problems. Section 4 closes the paper with a short conclusion and recommendation.

## 2 Requirements Analysis

As said before, process management is good for the support of complex application systems. We assume certain complexity since without that the usage of process management might be too much of a good thing. It is like with database systems: if just 50 to 100 data items have to be managed it might not to be recommended to deploy a database system. The effort and the cost that would have to be put into the development of a database application would not pay off for such a small application. The real benefit of database systems appears when a huge number of data items have to be managed in a very flexible way.

The comparison of process management with database management shows some interesting parallelisms. Therefore, we will analyze the area of process management by leaning on database management. Also, we will distinguish two issues when dealing with data and processes, respectively. First, things are defined; then, these things are used. Hence, two major tasks of database management are "data definition" and "data usage". The former means to define a database schema, the latter means to work with the data that are stored in a database system (inserting, modifying, deleting, and reading them). Analogous, we identify the two tasks "process definition" and "process usage". Process definition means to define process models that show how (process oriented) applications are structured. Process usage usually aims at the enactment of processes, i.e. processes have to be applied or used in applications. Often, process enactment is equal to "process execution", i.e. a system is (more or less) strictly executing a process step by step. However, we will discuss that this perspective is rather too narrow. To analyze both process modeling and process usage, we compare these tasks to the corresponding tasks in data management.

### 2.1 Process Modeling

First, the topics data definition and process definition, respectively, are discussed. There are many ways to define data structures for database applications. ER modeling is one of the most popular ones [15] and therefore we want to concentrate on it. Without going into the discussion of "unconventional" applications, the ER modeling method has proved its worth. This statement holds for "conventional" applications like in administration, production management, banking and insurance applications. We are well aware that ER approach shows limitation when "unconventional" applications like CAD or scientific applications (e.g. modeling a DNA) has to be enacted. Nevertheless, the comparison of conventional database applications is good enough to encounter requirements for process management applications.

What are the characteristics of conventional applications and why does ER modeling fulfill their requirements? Applications are characterized by data. It is necessary to define them first and then to define relationships between them. Data and relationships have to be characterized by certain individual features. All this is possible by applying the ER method.

The strength of ER modeling lies in its simplicity. There are two major concepts: entities and relationships. Entities describe data and relationships show how they are depending on each other. Due to these simple concepts almost any arbitrary data and relationships can be defined.

Due to the elementary character of the ER modeling concepts, almost any data structure can be defined. Most notably, there is no prescription how to model application data. For instance, it is not prescribed how to model a data structure for project management. For one application, there might be a project entity having 20 attributes; for another application, the project entity might have 30 attributes, etc. The relationship between projects and project managers might be n:1 in the first application and n:m in the second application. Nobody would buy a database system with a fixed project table, showing certain "burned-in" attributes and relationships. This would be like a "universal database scheme" for all applications which would not be acceptable of course. It might even look ridiculous to discuss this feature here. Nevertheless, this freedom with respect to modeling data structures is the key benefit of database systems. This observation becomes obvious when we look into the area of process modeling.

In the area of process modeling there is no consensus on what information to include into a process model, e.g. for the Travel Claim Reimbursement Process. (By the way, a process model is a complex data structure defining how process participants - e.g. process steps, documents, tools - are related to each other.) As a consequence, an all-embracing process modeling language does not exist. At a first glance, this is not a problem. It is comparable to the fact that database systems do not have predefined "burned-in" tables for projects etc. However, database systems offer elementary modeling primitives to construct individual, application specific tables for any application, i.e. also for project descriptions. Yet, this capability is not given for process management systems (i.e. for process modeling languages). These languages do not offer modeling primitives to define individual process models. Because they don't do it, it might again be interesting to have an all-embracing process modeling language by which any process model can be derived. Now, the lack of it becomes an issue indeed.

Consequently, the main question now is whether it is more appropriate

- to offer an all-embracing process modeling language or
- to offer elementary modeling primitives (by a process modeling language) for the definition of arbitrary process models?

Let us first pursue the issue of an all-embracing process modeling language. From a historical perspective there are approaches like CPM (Critical Path Method, [2]) or PERT (Program Evaluation and Review Technique, [1]) that are suggesting what to include into a process model. However, there are many more approaches and not all of them are compatible with each other. These days, the BPMN standard (Version 1.2 [5], Version 2.0 [6]) becomes very popular, although it is not applicable to arbitrary applications [19]. So, what to include into a standard process model?

All the proposals for a common process modeling language mentioned above are not "complete", i.e. there are features which are not supported. For example, the CPM method does not allow modeling the organizational perspective of a process, i.e. organizational policies which determine who should execute a process. Other methods do not support well the modeling of data or applications that are needed to describe a process step (e.g. BPMN, before Version 2.0).

In principle all these approaches assume a certain template for processes which comprises a concrete, pre-determined set of concepts. However, these templates are not "generally applicable" but are all developed from a certain perspective and for a certain application domain. Thus, they comprise certain process features, and neglect others. In contrast, different application domains normally require different modeling features. Two examples support this thesis.

- The product development process of a car manufacturer must be modeled. One major issue here is to exactly model persons that are responsible for the execution of process steps. In order to do so, powerful organizational relationships between responsible persons must be expressed. For example, it must be defined that a part release step must be executed by the project manager who is responsible for the design of the car the part is made for. This latter policy assignment is a rather tricky one.
- In a health care application it might be required to model the legal regulations for medical treatments. This means that corresponding process steps must be associated with those legal regulations, i.e. there must be a link attribute between process steps and legal regulations.

Obviously, the car manufacturer and the healthcare application request different process modeling languages. We do not want to criticize any of these approaches mentioned above (e.g. BPMN, CPM). They might be very valuable for the application domain they are stemming from and they are made for. However, we criticize when providers of those approaches claim to have "the" generally applicable process modeling template. This is definitely not the fact and it is not possible since not all features of all applications can be incorporated into such a process modeling language. Therefore, we see the approach of having a "one-fits-all" process modeling language as not achievable. Consequently, we see the adoption of the data modeling approach as much more eligible. This means that a set of elementary process modeling primitives should be defined. Those primitives can be applied by domain experts to construct their customized process models comprising all features a process must show from their perspective.

As a summary for the requirements analysis for process modeling we identify the following: The definition of process modeling primitives is required that can be used to construct domain specific process models. This approach works similar as the ER data modeling approach; there the data modeling primitives are sufficient to define individual domain specific data structures.

### 2.2 Process Execution

Second, the topics data usage and process usage, respectively, are discussed. Before data can be used the database has to be prepared for it, i.e. data definition must take

place and data structures must be defined. To simplify the discussion and concentrate on the essential, there are two principle usages of data: data retrieval and data modification. Data retrieval is very clear: data are read and presented. Data modification comprises inserting, modifying and deleting data.

Nevertheless, not in all databases data modification occurs according to the same pattern, i.e. different database applications need different kinds of data modification operations. We want to differentiate between "normal" databases and data warehouses – i.e. OLTP vs. OLAP databases [4]. In normal databases an update operation is one of the normal operations: one or many tuples have to be updated. In contrast, in data warehouses update operations are more crucial. Normally, data warehouses are "append-only" databases: new tuples are inserted; existing tuples are not changed. Only in cases of errors (e.g. wrong data are corrected or missing data are completed) tuples are updated. While there are no special follow-on operations necessary in databases, in data warehouses complex and complicated follow-on operations are required. Typically, pre-aggregated data marts have to be re-compiled. Since these are very cumbersome tasks optimizations should be applied [3].

A similar observation holds for data retrieval. While data are normally retrieved from base tables in databases, in data warehouses pre-aggregations (e.g. in form of data marts) are necessary to efficiently process the queries. Additionally, while normal databases applications are mostly working on base data, data warehouse applications work on aggregated data.

In summary, we see that the usage of data is quite different in databases. It is so diverse that even two different implementation concepts are necessary (normal databases vs. data warehouses). The same observation holds for process management. Not all kinds of usage are the same. We will discuss this observation in the following.

After having modeled processes they can be used. What does that mean? We shortly want to present the recent development of the process management area and want to show that this development led into a wrong direction.

It was at the beginning of the nineties. Process management became very popular, especially due to Hammer's and Champy's publication [13]. In reaction to that, a huge number of process execution systems evolved. They were mostly called workflow management systems. Their common characteristic is that processes are strictly executed according to their definition, i.e. according to the underlying process model. What does that mean? Process models describe a certain order of process steps. A workflow management system then takes the process definition and interprets it strictly, i.e. one step after the other is executed. Experiences with that way of process execution were very disappointing. Mostly it was criticized that workflow management systems are too inflexible and therefore not applicable. We can follow this argument since we are convinced that this kind of process execution is much too restrictive. Nevertheless, it might be absolutely adequate in certain application scenarios. In order to argue that the strict execution semantics of workflow management systems is not generally applicable we illustrate how processes are executed in real life.

For example, we look into an administrational application. In order to prepare a meeting, a manager's secretary is writing its agenda and an invitation letter which includes the list of people who should be invited to that meeting. This list strongly depends on the topics on the agenda. The invitation letter has to be signed by the

manager. The invitation letter should not be signed by the manager before both the agenda and the list of participants is complete since he is responsible for them. In principle the corresponding process looks as shown in Fig. 1:

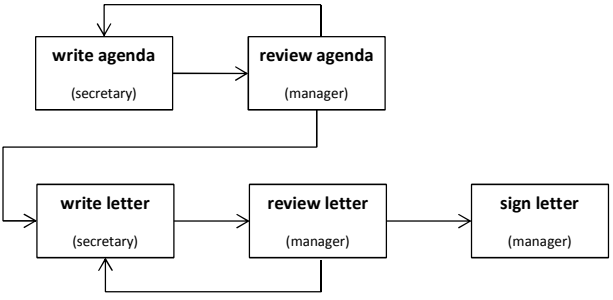


**Fig. 1.** Sample process

The process model in Fig. 1 seems to be correct. However, it is not practicable at all. In real life it will not be executed as shown in the figure since that would be too restrictive. The following situations might happen:

- After the manager has reviewed positively the letter or the agenda, one of these has to be changed again;
- After the manager has signed the letter, it must be corrected.

These are only two out of many "exceptional" executions that might happen. The question arises: why have those execution flows not been included into the process model before. We want to mention two reasons for that:

- The two above mentioned "exceptional" flows through the process are not the only ones. More are conceivable. To model all of them would increase drastically the complexity of the process model. Therefore, they are neglected;
- The two above mentioned "exceptional" flows are not modeled since they are not preferred and therefore they should not be offered.

Interpreting all of these arguments encounters a dilemma: on the one side, it cannot be avoided that "exceptional" executions like the above mentioned are relevant. If they are not modeled the process is much too restrictive and will most probably not be accepted by the users. On the other side, those executions should not be modeled since they are not recommended. Besides, to model all of them would enormously blow up the process model which then is not readable any more. How to resolve this dilemma?

Some consequences or requirements can be derived from the above observation:

- It is necessary to model "recommended" and "not recommended but nevertheless possible" paths in a process model. This is the only way to offer execution flexibility;

- In order to model these different kinds of execution paths, new modeling constructs are necessary. Just to use the conventional ones would make a process model unreadable.

As conventional modeling constructs we regard sequence, alternative and parallel executions and loops. These are the constructs which are borrowed from programming languages. Advanced modeling constructs also are able to model – among other things – recommended and exceptional paths.

To introduce new process modeling constructs defines new tasks for both process modeling and process execution. Therefore, we discuss this issue both in Section 3.2 and Section 3.3, respectively.

Another consequence of the scenario above should be considered that cannot be directly derived from the example. Why to restrict process execution to the way workflow management systems are doing it? Are there alternative ways of execution, i.e. usage of a process model? We think so. In Section 3.3 we will present such alternative interpretations.

## 3 Supporting Domain Specific Process Modeling and Execution

In Section 2.1 a few requirements for process modeling are identified. First, we look for modeling primitives. These modeling primitives can be used to define customized modeling constructs. Additionally, in Section 2.2 the need for new modeling constructs with new semantics has been identified. They are needed to express execution semantics beyond the usual execution semantics of workflow management systems. In this section we demonstrate how all these requirements can be met by a multi layer meta modeling approach. Presenting this approach does not mean that it is the optimal one; it just acts as a proof of concept. In the cited papers related approaches are discussed; this discussion should not be repeated here.

### 3.1 Process Modeling Primitives

At first, the issue of identifying modeling primitives is tackled. Therefore, we present the POPM (Perspective Oriented Process Modeling) approach; it is capable of offering process modeling primitives.

The POPM approach was first presented about 15 years ago in [16] and [17]; in [18] its conceptual backbone is reconstructed in detail. We refer to this publication throughout the following discussion.

The fundamental idea of POPM is not to offer process modeling constructs directly, but to offer a method for defining process modeling constructs. The enactment of the POPM method relies on a so called meta model stack. The basic idea of a meta model stack is the following: on level x (abstract) modeling elements are offered that can be used at level x-1 to define (concrete) modeling constructs. This approach resembles the rationale of MOF (Meta Object Facility, [5]). However, our approach heavily leverages on advanced modeling concepts like powertypes, deep instantiation, materialization, and clabjects which are not supported by the MOF approach (see [18] for details). Fig. 2 presents the POPM approach at a glance:

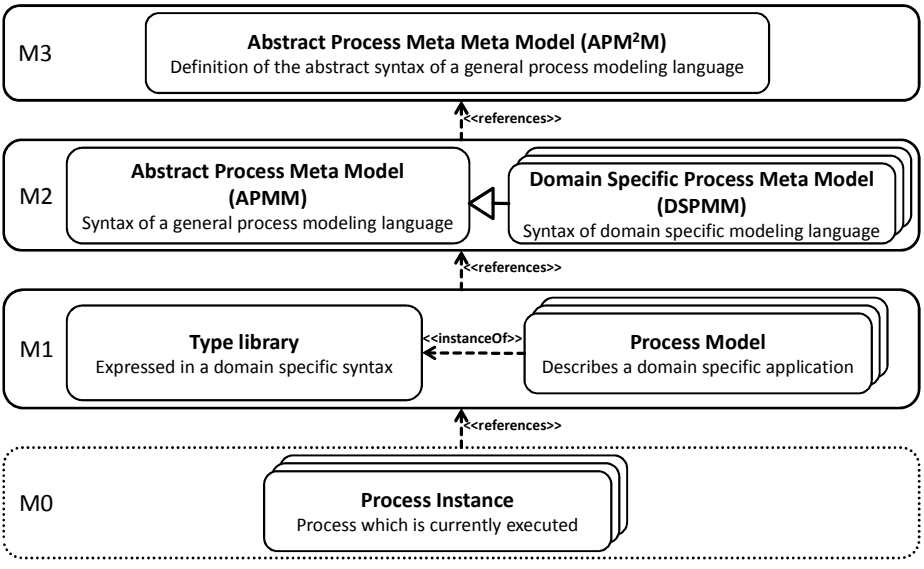


**Fig. 2.** Meta layer stack of POPM

Process models are defined on the modeling layer M1 (right boxes). A process model uses process (and data, organization etc.) modeling constructs which are accumulated in the "Type library" on M1 (left box). These modeling constructs are provided domain specifically; they are defined on the next upper level M2.

All process definitions on M1 are defined using the process modeling constructs offered by a process modeling language which is specified at level M2. Such a process modeling language is defined as a so called Domain Specific Process Meta Model (DSPMM). M2 further contains the definition of an abstract process meta model (APMM) defining a set of general language features for processes, data flow or control flow. These features are relevant for almost all process modeling languages; therefore they are collected in the APMM. All modeling elements on level M2 are process modeling primitives: They can be combined arbitrarily to set up domain specific process modeling languages. For instance, process modeling languages for the two examples in Section 2.1 (car manufacturer, health care application) can be defined. Each DSPMM defines an individual (we often say: domain specific) process modeling language (DSL: domain specific language). Thus, the above mentioned requirement is fulfilled.

Fig. 2 shows two further levels: M3 and M0. Level M3 offers generic modeling elements (boxes, arcs, etc.) which are used to define modeling primitives on level M2. Further, level M0 contains running instances of processes defined on M1 (right boxes).

It is not in the scope of this paper to prove that the modeling constructs offered on level M2 allow defining arbitrary process modeling constructs. We rather want to refer to [10] and [11] that describe various applications which all require specific process modeling languages; all of them could be provided by the POPM approach.

As a first resume, we can state that it is possible to offer individual process modeling constructs as requested in Section 2.1. Therefore, the lack of a one-fits-all standard process modeling language is not so critical. Especially, it does not prevent process modeling to be a broadly applicable method.

### 3.2 Domain Specific Process Modeling Constructs

Since the POPM approach supports the definition of new modeling constructs (assembled out of modeling primitives offered by layer M2) it is possible to define process modeling constructs with customized, individual semantics. This feature can be utilized to model compact modeling constructs or modeling constructs with specific execution semantics. These modeling constructs can be used, among other things, to support scenarios as presented in Section 2.2 (recommended execution paths). Nevertheless, just to offer new modeling constructs is not sufficient. Also new execution semantics must be provided. This will be done in Section 3.3.

Why introduce new process modeling constructs? There are principally two reasons:

- To represent certain scenarios with conventional modeling constructs would lead to unreadable and incomprehensible process models. Thus, new (e.g. compact) process modeling constructs are defined that model complex situations with plain means (i.e. the resulting process models are easy to read);
- In some scenarios specific execution semantics are necessary. To express this, new modeling constructs are needed.

Again as a proof of concept we introduce a piece of research work that successfully is coping with the two issues above. It is introduced in [19] and is called ESProNa. The ESProNa approach aims at process based applications with huge variability. Again, this is not to say that this research is best, it is just to present a successful feasibility study. Related approaches are discussed in the cited literature.

In [19] three new modeling constructs are introduced: two of them, namely two kinds of arrows will be presented in the following. The challenge for ESProNa is to present process applications as presented in Fig. 1 in an easy, readable way. Such process applications are characterized by showing a huge number of alternative execution paths. Although all of them should be offered, some of them should explicitly be marked as recommended; others should be marked as exceptional.

The following discussion only focuses on the control flow perspective of a process model, which this is absolutely sufficient for the purpose of this paper. ESProNa basically introduces two special process modeling constructs for control flow. The first process modeling construct is the normal arrow. The semantic of the well known arrow symbol in process modeling is that if an arrow directs from process A to process B then process B has to be performed after process A. Accordingly, if process B is connected with an arrow to process C then C may start after process B has finished. We also say: B requires the execution of A before it can run; C requires the execution of B (and consequently of A, too) before it can run. This arrow is represented by solid lines since it strictly prescribes an execution order (Fig. 3).

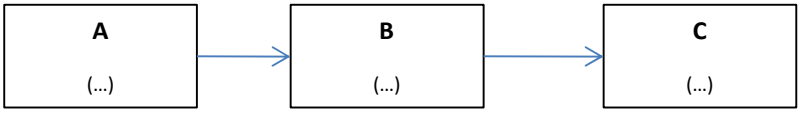


**Fig. 3.** The solid arrow

Beside this arrow construct depicted by a solid line we want to add an arrow depicted by a dashed line; this dashed arrow holds a different meaning. Two processes that are connected through a dashed arrow can – in principle – be executed in any order. For instance, if process A and process B are connected by a dashed arrow A can be performed before B or vice versa, B can be performed before A. Nevertheless, having defined a dashed arrow from process A to process B expresses a preference (recommendation) that process A should be performed before process B. This feature can be utilized when processes are put on a work list for execution. If more than two processes are connected through a dashed line then a permutation of all process executions is feasible, e.g. ABC, BCA, CBA (Fig. 4).

The introduction of the dashed arrow is a fundamental step with respect to the expressiveness of a process model. The interpretation of solid arrows is borrowed from classical logics [24]. There the "tertium non datur" holds; this is the principle of the excluded third. That means that if a proposition S holds (it is true) the inverse proposition ¬S does not hold (it is false); there are no other states than these two. In contrast, the interpretation of the dashed arrows more stems from constructive logics [22] [23]. There, only the principle of the "exclusion of contradictions" holds which is less strict than the "tertium non datur". It means that everything holds but those things that are explicitly excluded; in other words, no proposition can be true and false at the same time.

Also this is just a short and informal discussion it already reveals that by applying concepts of the constructive logics a much wider spectrum of execution paths result from a process model. The step from classical to constructive logics could be compared with the step from algorithm based computing to interactive computing that was discussed by Wegener [21]. It is not that this research work is directly comparable with our ideas it is just to say that a similar kind of design rationale lies behind these two research approaches.

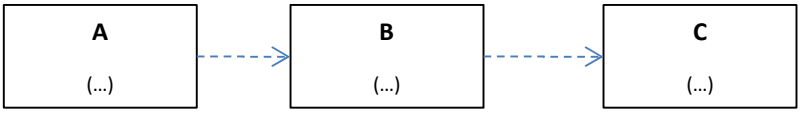


**Fig. 4.** The dashed arrow

It is certainly possible to combine the solid and dashed arrows. For example, process A and B are connected through a dashed arrow; process B and process C are connected through a solid arrow. This means that there is flexible ordering between processes A and B while process B must always be executed before process C. This semantics results in the following three execution orders: ABC, BAC and BCA (Fig. 5).

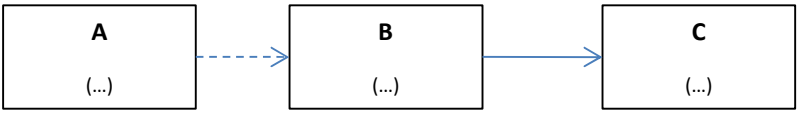


**Fig. 5.** Combination of dashed and strict arrow

Of course it is not enough that only new process modeling constructs are defined. Also an adequate execution infrastructure must be made available. ESProNa supports such an infrastructure. We will discuss this issue in Section 3.3.

Having available these two new process modeling constructs the example of Fig. 1 can be modeled as depicted in Fig. 6.

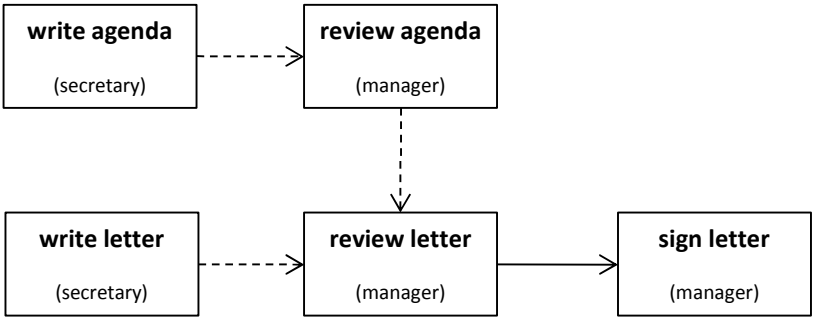


**Fig. 6.** The example of Fig. 1 – remodeled

At a first glance the revised process model does not look much simpler than the original process model from Fig. 1. However, we could argue that we have reduced the number of arrows from six to four. Although, this 33% reduction is a phenomena of the "statistics of small numbers" there is some truth with it. The process model from Fig. 6 is much more powerful than the process model from Fig. 1. Especially, all "exceptions" reported in Section 2.1 are covered by this revised process model. Additionally, the recommended path clearly appears in Fig. 6. The latter two arguments are the decisive ones. Besides, the process model is more readable. The attentive reader might have recognized that in the remodeled process also a review step can be performed before the corresponding write step is done. This is definitely intended: sometimes people want to bring forward a specific piece of work although they know that they cannot complete it. Nevertheless, they want to pre-executed the step since they want to adhere to a specific information in order not to forget it afterwards when they really complete the step.

In summary, we want to note that new, domain specific modeling constructs have the ability to model complex application scenarios with compact, well-arranged process models. Though, the process management community is requested to put more research into this issue since process models are the communication means between domain experts and (technical) process experts.

At the end of this section about domain specific process modeling we want to discuss alternative approaches. Of course, UML as the classical modeling language must be discussed. Since pure UML does not offer constructs that are sufficient for process modeling we look at extensibility mechanism of UML. The UML provides an extension mechanism that allows for creating Domain Specific Languages as well. Extensions are encapsulated within so-called Profiles which can be loaded into UML modeling tools [8][9]. These Profiles may not only specify language constructs but also a (graphical) notation for these constructs. However, with Profiles only such adaptations can be performed which do not alter or break the semantic of constructs contained within the UML. Therefore, language features that harm basic concepts of the UML such as Clabjects or Deep Instantiation cannot be put into a profile. Under the assumption that Clabjects and Deep Instantiation are fundamental for our approach, UML Profiles could not be taken to fully implement it.

### 3.3 Supporting Process Execution

In Section 2.2 the strict execution of processes through workflow management systems was criticized. Besides, it was motivated to think about alternative execution semantics. Thus, we want to go more into the direction of "process usage" than "process execution" – the latter always more or less directly directs to computer based enactment. Nevertheless, also computer based enactment strategies can provide flexibility that is needed by process users. In the following discussion we will stepwise "macerate" the strict execution semantics of workflow management systems.

We do regard the execution semantics as one very valuable interpretation of "process usage". There are application scenarios where this very strict execution strategy fits well. For example, when new construction drawings, CAD models, etc. have to be released within a design process, this has to be done according to fixed and strict rules. Responsible people have to release these artifacts, the latter have to be registered in corresponding repositories, and finally they have to be stored at predefined and predetermined locations. Each step within such a process must occur exactly in the right order. Thus, a workflow management system can perfectly support such a scenario.

However, it is obvious that the workflow management support is not suitable for other phases of a design process. For instance, in the more creative phases of a design process, no one wants to be strictly guided by a workflow management system. Rather, designers want to be free in their decision what to do next since design is a creative and non-predictable process. Nevertheless, designers would like to be disburdened by a system that helps them to find the right documents, tools, etc. when they are working on a specific step in the design process. For instance, when they want to do an FEM computation they want to be provided with all relevant documents (e.g. CAD model) and want to get offered the adequate tool for this computation. Also, they want to see whether all relevant documents are already available and might even want to get an overview about former occurrences of this design situation. Novice designers also are content when they are accommodated with an overview of the design process such that they get an idea what process steps to do next. Altogether, we call a system that supports all these mentioned issues a process navigator: it shows where a user stands in the process, depicts what resources (documents, tools, etc.) are

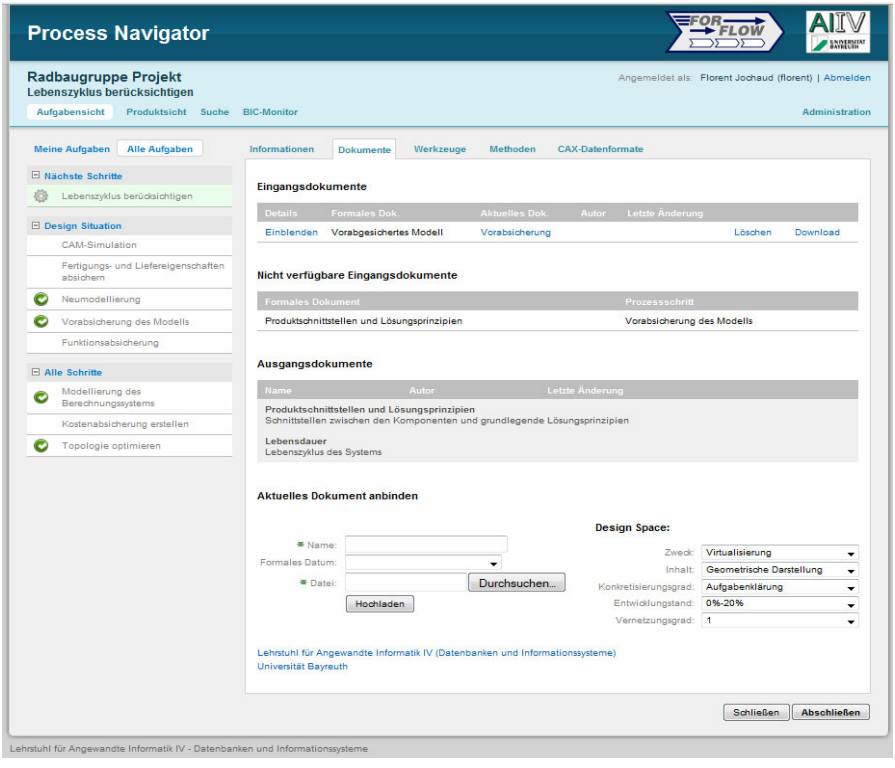


**Fig. 7.** The ProcessNavigator

currently available or needed, provides an outlook on what to do next and even allows browsing the history of former process executions. Again, as a proof of concept we will introduce a system that supports this special kind of process support: it is called ProcessNavigator [20]. We shortly present the ProcessNavigator in the following in terms of presenting a feasibility study, i.e. a systems that copes with the requirements posted so far.

The ProcessNavigator is depicted in Fig. 7. The left part of the ProcessNavigator provides different sorts of worklists. The upper worklist determines the process steps that – according to an underlying process model – should be executed next. However, the second worklists offers process steps that are – most probably – also relevant for the current situation. A third worklist acts as fall back and offers all process steps available in the process model. The user interprets the three worklists as follows: in the first worklists the highly recommended process steps are listed; the second worklist offers process steps that are fine but not really recommended. In order to cope with more exceptional situations the process steps from the third worklist can be selected.

Through the separation of different worklists the ProcessNavigator offers both guidance and flexibility for process execution. It is interesting to know that the ProcessNavigator is based upon the ESProNa method. Through its logic based implementation

not only recommendations about process steps can be given, also the ProcessNavigator can provide hints about effects of process executions. For example, if the process step "sign letter" is performed, no other process steps can further be executed. This functionality allows executing so-called what-if-games and again helping users to better navigate through a process.

The right part of the ProcessNavigator provides situation specific information. Among other things, it is depicted what documents are available for the execution of a specific process step, what tools could be applied, etc. Besides, historical data, i.e. data about former executions of the process can be retrieved. Altogether this information helps users to easier, faster and more consistently execute process steps.

Both the workflow execution mode and the ProcessNavigator execution mode are closely oriented to the underlying process model. Also, both execution modes take a process model and offer a user interface that allows a user to navigate – with more or less freedom – through a process model. In a project in the administration department of a university we found that alternative execution modes are also relevant. We call this new execution mode checklist method.

A second example once more stems from an administrational application. Again, a large process model is defined (> 250 process steps). Again, the variability of execution is huge. And what is more important, most of the process steps are so called manual tasks. We call a process step a manual task when it has to be worked upon mostly without computer support. For example, a paper document has to be signed and has to be sent to a following station. Nevertheless, the domain experts want to somehow be able to track and supervise process executions. Therefore, they developed the following concept.

First, all processes are modeled as usual. Then, for each process a so called checklist was defined. A checklist comprises the main process steps including documents that must be produced and agents that are responsible to perform the corresponding process. Agents had to sign it (electronically) when they have finished a certain process step. Through this method it was possible to track the execution of a process although most parts of the process were performed external to a computer system.

| **Main Process** | | | |
|---|---|---|---|
| **Process**<br>A | **Documents**:<br>**IN**: Doc 1<br>**OUT**: Doc 2 | **Comment**: | **Agent**: Y<br><br>**Signature**: _____ |
| **Process**<br>B | **Documents**:<br>**IN**: Doc 3<br>**IN**: Doc 4 | **Comment**: | **Agent**: X<br><br>**Signature**: _____ |

**Fig. 8.** Checklist

Fig. 8 depicts the principle structure of a sample checklist. It serializes the process steps of a process model and additionally shows what input sources could be used (Documents "IN"), what results are expected (Documents "OUT"), comments on the execution (Comment; that is very important to track experience), and who is eligible to perform the process steps (Agent). A signature – it can be either electronic or paper

based – confirms the execution of a process step. It has to be mentioned that the checklist method offers even more flexibility as the ProcessNavigator. In principle the one important statement is that at the end of the process all signatures are on the checklist. So it is completely output oriented. Nevertheless, the checklist method describes a very interesting and valuable form of process usage and widens its spectrum towards non-computer based and extremely flexible process execution.

Last but not least we want to introduce another form of process usage. It completely neglects computer support for process execution but only relies on process modeling. We motivate this approach with an example.

In a project with a sports and fashion company the production process for shoes has to be defined. The outcome of the modeling phase is a large process model comprising more than 200 process steps. Up to this point the project is absolutely conventional in the course of project management, i.e. conventional process modeling is encountered. However, the process "usage" is quite different to conventional process "execution". The company is recognizing that although the modeled production process is correct as a template, each concrete production process (for each production lot) is special and is deviating in many small places from the template process. Due to this variability it is not possible to model all these variants. Although there was no need to apply a tool like the ProcessNavigator since this kind of direct project control is not required. For the process is mainly used by the central headquarter department to supervise the process on an abstract level. Nevertheless, the company wants to make use of the production process model. Therefore, they choose the following kind of execution: The production process model is printed out whereby the process steps are arranged in a special way. Then this process model is stuck onto a wall as wall paper. With little colored flags the managers are indicating the progress of each concrete production process instance. So, deviations can easily be tackled with since just the little flags had to be rearranged accordingly.

The company is profiting enormously from this unconventional way of process usage since they always can keep an overview onto all production processes and could track and supervise them.

This example shows that process "execution" should more appropriately be interpreted as process "usage". It also shows that not only the complete automation of process execution is useful. We call this kind of process usage external tracking.

External tracking and the checklist method are two unconventional ways of executing process models. Nevertheless, they fully fulfill their purpose to take advantage from process models.

At the end of this section it should be mentioned that there is a whole bunch of approaches that aim at the improvement of flexibility in process execution. [20] discusses and classifies some of them. Additionally, there are powerful approach to cope with dynamic development processes [14]. Also these approaches deal with some special kind of flexibility, among other things the flexibility of dynamically changing processes. We do not want to ignore all these approaches; they are very valuable and improve the flexibility of (conventional) process execution. However, in this paper we do not want to discuss approaches for flexible process execution alone but want to show that there are alternatives to (conventional) process execution approaches. These mostly stem from the introduction of domain specific, compact process modeling constructs.

### 3.4 Classification

Although we have introduced new interpretations of process execution and have shown how they could be enacted, it must be mentioned that the conventional process execution modes as they are supported by workflow management systems is still a valid candidate. There are processes that require a strict execution order which must not be violated. In such cases conventional workflow management systems provide a valuable platform for process execution.

Altogether, this section should show that other process usages than running them on workflow management systems is feasible and relevant. It will typically be the case that in one application domain a mixture of all these process execution modes will be relevant. Although we have investigated all four usage modes in detail, the remaining challenge is to combine them in one comprehensive application system. This could also be a motivation for researchers in general to think about alternative execution modes for processes.

In summary we want to depict the four introduced execution modes on a scale; this shows a broad spectrum of execution modes and should trigger researchers to look for more in order to more adequately meet the requirements of process applications. Fig. 9 shows the spectrum of the usage modes presented in this section. We span two dimensions: a first dimension distinguishing between strictness and flexibility with respect to process execution; a second dimension distinguishing between usages within a computer system (internal) and outside of a computer system (external). By "internal" we mean that most of the applications and the usage of the system take place on a computer system; "external" means that many/some actions occur completely outside a computer system (e.g. the wallpaper approach).

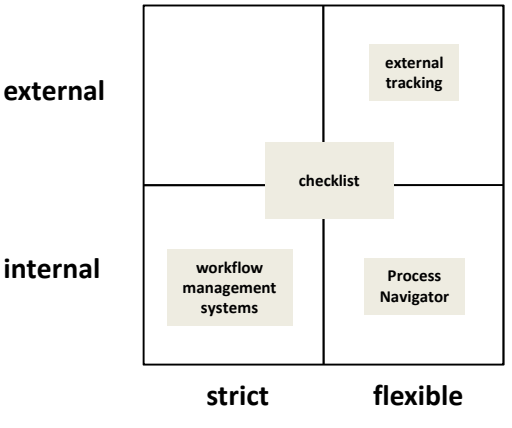


**Fig. 9.** Spectrum of process usage

Of course, workflow management systems are the handling process execution within a computer system and prescribe very strictly what to do. The ProcessNavigator relaxes the execution of processes but mainly aims at process executions within a computer system. The external tracking approach is most flexible and also leaves the

technical environment; however it is also possible to implement it on a computer system. The checklist approach can be configured both flexible and strict, and both intern and extern. So, it allows most degrees of freedom, depending on how it is implemented.

## 4 Conclusion and Outlook

At the end of this paper we want to summarize its main contribution. The purpose of this paper was not to introduce another piece of research. Rather, we wanted to stimulate the process management community to think about alternative and more powerful approaches to process modeling and execution.

Through presenting a couple of feasibility studies we already could show that there are valuable alternatives that are worth to be investigated further. We are convinced that these approaches are just the starting points for more appropriate ways of process management.

In detail, we want to motivate to contemplate the following research issues:

- Is there a set of elementary process modeling elements?
- How to develop domain specific models effectively and efficiently?
- How to find out which domain specific modeling primitives are optimal for a specific domain?
- What does "enactment" of a process model mean? Is it more than execution?
- How to make process execution more flexible?

These questions point to research issues which altogether aim at more adjustable and customizable process modeling language, since we regard adjustability and customizability as two of the key factors for the acceptance of process management.

## References

[1] Program Evaluation and Review Technique, Wikipedia, http://en.wikipedia.org/wiki/Program_Evaluation_and_Review_Technique (retrieved: 2009-11-03)

[2] Critical Path Method, Wikipedia, http://en.wikipedia.org/wiki/Critical_path_method (cited: 2009-11-03)

[3] Kimball, R., Ross, M., Thornthwaite, W.: Kimball's Data Warehouse Toolkit Classics. John Wiley & Sons, Chichester (2009)

[4] Elmasri, R., Navathe, S.: Fundamentals of Database Systems. Pearson Education, London (2006)

[5] Object Management Group, Business Process Model and Notation (BPMN), Version 1.2, OMG Document Number: formal/2009-01-03, http://www.omg.org/spec/BPMN/1.2 (cited: 2009-11-03)

[6] Object Management Group, Business Process Model and Notation (BPMN), FTF Beta 1 for Version 2.0, OMG Document Number: dtc/2009-08-14, http://www.omg.org/spec/BPMN/2.0 (cited: 2009-11-03)

[7] Object Management Group, MOF 2.0 Specification, http://www.omg.org/spec/MOF/2.0/ (cited: 2009-11-01)
[8] Object Management Group: UML 2.2 Super- & Infrastructure. In: OMG (ed.) (2009), http://www.omg.org/spec/UML/2.2/
[9] Faerber, M., Meerkamm, S., Jablonski, S.: The ProcessNavigator - Flexible Process Execution for Product Development Projects. In: International Conference on Engineering Design (ICED), Stanford, CA, USA, August 24-27 (2009)
[10] Faerber, M., Jochaud, F., Jablonski, S., Stöber, C., Meerkamm, H.: Knowledge oriented Process Design for DfX. In: 10th International Design Conference, Dubrovnik, May 15-18. The Design Society (2008)
[11] Faerber, M., Jablonski, S., Schneider, T.: A Comprehensive Modeling Language for Clinical Processes. In: 2nd European Conference on eHealth (ECEH 2007) (2007)
[12] Fuentes-Fernández, L., Vallecillo-Moreno, A.: An Introduction to UML Profiles. UPGRADE - The European Journal for the Informatics Professional 5, 6–13 (2004)
[13] Hammer, M., Champy, J.: Reengineering the Corporation. In: A Manifesto for Business Revolution. Nicholas Breadley, London (1993)
[14] Heer, T., Heller, M., Westfechtel, B., Woerzberger, R.: Tool Support for Dynamic Development Processes (to be published in this book)
[15] Chen, P.: The Entity Relationship Model – Toward a Unified View of Data. TODS, 1, 1 (1976)
[16] Jablonski, S.: MOBILE: A Modular Workflow Model and Architecture. In: Proc. International Working Conference on Dynamic Modelling and Information Systems, Noordwijkerhout, NL (1994)
[17] Jablonski, S., Bussler, C.: Workflow Management: Modeling Concepts, Architecture and Implementation. International Thomson (1996)
[18] Jablonski, S., Volz, B., Dornstauder, S.: On the Implementation of Tools for Domain Specific Process Modelling. In: 4th International Conference on the Evaluation of Novel Approaches to Software Engineering (ENASE), Milan, Italy, May 9-10 (2009)
[19] Jablonski, S., Igler, M., Günther, C.: Supporting Collaborative Work through Flexible Process Execution. In: The 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2009) (2009)
[20] Jablonski, S., van der Aalst, W.M.P.: Dealing with workflow change: identification of issues and solutions. International Journal of Computer Systems Science & Engineering 15(5) (2000)
[21] Wegener, P.: Why interaction is more powerful than algorithms. Commun. ACM 40, 5 (1997)
[22] Lorenzen, P.: Dialogical Foundation of Logical Calculi. Constructive Philosophy (1987)
[23] Hughes/ Cresswell: An Introduction to Modal Logic, Methuen (1982)
[24] Tarski, A., Helmer, O.: Introduction to Logic. Dover Pubn Inc., New York (1995)