Semantic Process Modeling and Planning

Michael Igler
Chair for Applied Computer Science IV
University of Bayreuth
Bayreuth, Germany
michael.igler@uni-bayreuth.de

Stefan Jablonski
Chair for Applied Computer Science IV
University of Bayreuth
Bayreuth, Germany
stefan.jablonski@uni-bayreuth.de

Christoph Günther
Chair for Applied Computer Science IV
University of Bayreuth
Bayreuth, Germany
christoph.guenther@uni-bayreuth.de

***Abstract*—In this paper, we investigate how complex process models can be modeled such that both the modeling remains doable for domain experts and the resulting process models remain readable. We chose an approach that can be characterized as mixed approach consisting of declarative and imperative modeling aspects with semantically enriched process modeling constructs. Our aim is to benefit from both modeling approaches, declarative and imperative, whereby through their combination we want to avoid their drawbacks. However, the implementation of our modeling approach is completely declarative. In this paper we present our novel approach for process modeling and together with its implementation. Some experiences about how domain users are applying this approach are also given.**

***Keywords*-Semantically enriched process modeling constructs; flexible process execution; process planning.**

I. Introduction

Process management has been accepted as adequate method to describe complex business applications and to support their enactment. Deliberately we focus on complex applications since there the benefits of a process-based approach are of particular importance. Process models illustrate nicely how complex applications are structured and describe what has to be done by which person using which tools. However, we believe that process management approaches still do not cope well with complexity. In order to substantiate this proposition we want to analyze the causes of complexity. We focus the discussion of complexity on two situations. A process-based application is complex if it consists of a huge number of different process steps (step complexity). It is not so easy to reduce this kind of complexity. Such an application can be structured by creating sub-processes through decomposition. Then process models are at least easier to understand. However, it is hard to eliminate process steps such that the application gets ”smaller”. Step complexity is a kind of an inherited feature. There is a chance that domain experts recognize that some process steps are not necessary; then this complexity can be reduced partially. A second sort of complexity arises when a huge number of execution paths exists (path complexity). In this case, the number of process steps might even be moderate. However, through the flexibility of many different execution paths complexity escalates. For example, consider three process steps `A`, `B`, and `C`

- which all have to be executed exactly once, and
- whose executions must not overlap.

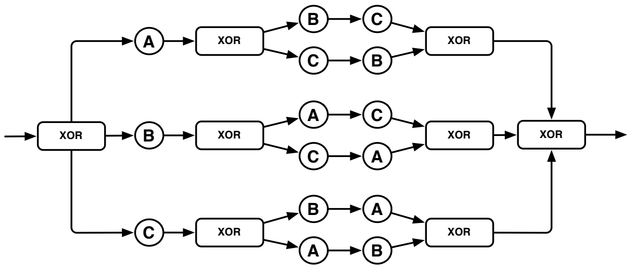

Fig. 1. Example process model

In Figure 1, a solution to this scenario is depicted. We regard this process model as complex: although only three different process steps are involved, the process model consists of 15 process steps (repetitions of the three basic steps `A`, `B`, and `C`), 29 arcs, and 8 flow constructs (XOR) for splitting and joining control flow. In this context it is not so relevant how to count steps and arcs; the message is that there are a lot of modeling elements although the application is rather small. The most severe drawback of this process model is that its pragmatics (what it means from an application point of view) is totally camouflaged, i.e., users do not comprehend the meaning and purpose of the process. We state that path complexity is partially avoidable when powerful process modeling constructs are applied.

II. Motivation

What is the reason that still path complexity is not dealt with adequately? We see one of the major reasons in the adoption of execution rules from imperative programming languages like sequential execution, alternative execution (if-then-else; XOR between execution paths) or independent execution (parallel execution paths). It is not that we blame imperative programming languages it is just that we state that this programming style is not adequate for process modeling. The fact that programs are going to become complex is not that bad since programs are just read by programmers, i.e., software experts that are able to cope with that complexity. In contrast to that process model complexity is problematic. Process models must also (besides professional process modelers) be readable for end users like medical doctors or nurses, who are usually not so familiar with formal process modeling techniques. Thus, when process models are becoming too complex, these people cannot interpret them anymore. That also means that they cannot assess their quality anymore and therefore cannot improve them. As a consequence we really want to promote applying process modeling techniques, which reduces complexity such that complex applications can be described by comprehendible process models and can therefore be understood much more easily. We propose to apply declarative process modeling techniques that specifically reduce path complexity. In contrast to imperative modeling (here the path(s) going through a process is defined explicitly) declarative modeling concentrates on describing what has to be done and the exact step-by-step execution order is not directly prescribed.

In order to provide another motivating example we introduce a second scenario which would result in a most complex process model if it is only described with conventional process modeling elements (such that are borrowed from imperative programming). The (simplified) scenario originates from a clinical study. Four process steps are involved: `Take blood sample` (from now on called `A`), `Measure intraocular pressure (B)`, `Measure blood pressure (C)`, and `Write report (D)`. It is not a problem to model this scenario with conventional means under the assumption that the four steps should be executed sequentially and each step must be executed exactly once. However, if it is allowed to execute `A`, `B`, and `C` in any order (but not overlapping) then the model adopts the complexity of the process model in Figure 1 and is presented in Figure 6. Complexity increases again when process steps `A`, `B`, and `C` could be executed multiple times (if their execution results are not satisfactory). Finally we would like to restrict the number of executions for each process step `A`, `B`, and `C` individually. Process step `A` should be executed once or twice, process step `B` doesnt have to be executed at all but can be performed once; `C` must be executed once and can be repeated arbitrarily. We abstain from listing the possible execution sequences. It becomes obvious that a huge number of such sequences could be produced. It is also a challenging exercise to define process models for the two last extensions of our application scenario.

Our goal is to introduce process modeling constructs that facilitate to model complex and flexible processes in a compact and comprehensible way. Our first idea was to switch from imperative modeling (as it is quite common in process management) to declarative modeling (see Section IV). However, studying [1], which discusses the pros and cons of imperative and declarative process modeling, we decided that a complete switch to declarative process modeling is not optimal, since that would mean to abandon the good aspects of imperative modeling. Thats why we have chosen an approach that tries to combine the pros of both approaches: declarative and imperative. Thus, we can reduce the obstacles of both modeling approaches as well. So, our modeling approach is a combination of declarative and imperative concepts. However, the underlying implementation of our process execution engine is a purely declarative one.

Besides this main requirement we want to post two further requirements, which are vital for our approach. We state that the semantics of these new process modeling constructs must be unique, i.e., they must be precisely defined. This requirement refers to both process modeling and execution. We explicitly name this requirement since we will allow a great degree of freedom for process execution. This looks like allowing arbitrary execution orders. However, this is not the case as we will discuss in Section IV. The third major requirement focuses process execution. Since we deal with complex scenarios that should be described by compact process models, end users should be guided through the execution of such processes, i.e., there should be a possibility to highlight recommended execution paths among the many eligible execution paths. Hence we give the process executors (e. g. nurse, medical doctor) some guidance through the process flow but still sustain the flexibility in choosing the next process step according to the actual personal perception of the process executors involved.

III. Related Work

The current approaches to process modeling can be categorized as either imperative approaches or declarative approaches. In this section, we compare our work with some representative implementations of both imperative and declarative approaches.

A. Imperative approaches

Wohed et al. [2] made an evaluation of the suitability of the imperative modeling language BPMN [3]. It evaluates the modeling languages against the workflow patterns from [4] concluding that there exist inherent difficulties in applying a language that does not have commonly agreed-upon formal semantics nor an execution environment. Although there is a mapping from BPMN to the execution environment BPEL [5], closer inspections show that this mapping is only partial, leaving aside models with unstructured topologies.

The research described in [6] does a comparison of business modeling and execution languages coming from the open source area. It concludes that open source systems like jBPM [7] and OpenWFE (now called ruote 2.1) [8] are geared more towards modelers who are familiar in programming languages than towards business analysts.

YAWL (Yet Another Workflow Language) [9] is a workflow language that make use of so called high-level Petri nets to refer to Petri nets extended with color, time and hierarchy. The definition of YAWL presented in [9] only supports the control-flow (behavioral) perspective. Therefore newYAWL [10] has been developed to provide support for the control-flow, data and resource perspectives. Nevertheless the functional and organizational perspective are still neglected.

B. Declarative approaches

DECLARE [11] is a constraint-based system, developed at the University of Eindhoven, that is focused on modeling constraints between processes. It supports the behavioral and the functional perspectives of Perspective Oriented Process Modeling method (POPM) [12]. DECLARE uses the ConDec [13] modeling language. Modeled constraints in ConDec are translated to a Linear Temporal Logic (LTL) formula. There is an automaton generated for every specific constraint in order to verify it. Furthermore, an automaton is also generated over all constraints. The support for the organizational perspective in DECLARE is, however, limited as hierarchical structures cannot be modeled. A planning component that can be consulted for advice during execution phase is also absent.

EM-BrA^{2}CE (Enterprise Modeling using Business Rules, Agents, Activities, Concepts and Events) is a framework for unifying vocabulary and execution models for declarative process modeling [14]. The vocabulary is described in terms of the Semantics for Business Vocabulary and Rules (SBVR) standard and the execution model is presented as a Colored Petri Net (CP-Net). EM-BrA^{2}CE also follows the same concept we use in this paper to specify a state space transition relation based on rules. However, functional and operational perspectives are not supported in this framework. Furthermore, the process modeler has no possibility to graphically “model” business process. Instead, every process must be described in the form of the mentioned Business Vocabulary. This slows down re-reading of process models by different users or the process modeler itself after some period of time.

IV. Introducing New Elements for Compact Process Modeling

This section presents three new modeling elements, which form the basis of our approach. Since we focus on the reduction of path complexity we introduce three new declarative modeling elements: special arrows (with two different semantics), boxes (to group processes), and quantification (to define the number of executions of a process). Besides these new modeling constructs we rely on the typical modeling elements of the perspective oriented process modeling method [15]. However, in this paper we mainly focus on the functional perspective and the behavioral perspective, whereas we neglect the data, operational and organizational perspectives.

A. Two Different Types of Arrows

The first modeling construct that will be associated with a new semantics is the arrow. The semantic of the well known arrow symbol in process modeling is that if an arrow connects process `A` with 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 (Figure 2). 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. We want to keep this very common construct and put it in our modeling toolbox. We present this modeling construct as a solid line.

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

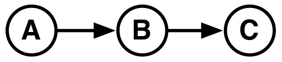

Fig. 2. Sequential process flow

through a dashed arrow can be executed in any order. For instance, if process `A` and process `B` are connected by a dashed arrow, then `A` can be performed before `B` or vice versa (`B` can be performed before `A`). Along with this construct we see the optimal combination of the imperative and the declarative approaches. Applying imperative concepts would require much more arrows and deciders to express the flexibility; this would blow up the process model drastically and lead to unreadable models; exclusively applying declarative concepts would avoid any arrows and the ”natural” understanding of a process flow would be lost. Furthermore, 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`. Processes must not be performed in parallel but sequentially. Modeling the scenario from Figure 1 using the dashed arrow, results in the simple process model of Figure 3. It is obvious that the process model of Figure 3 is much more readable than the process model of Figure 1.

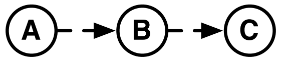

Fig. 3. Model of flexible scenario

It is certainly possible to combine the solid and dashed arrows: In Figure 4 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`.

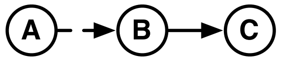

Fig. 4. Combination of solid and dashed arrows

Now consider the medical example from Section I. To connect the three process steps `A`, `B`, and `C` with dashed arrows offers to execute them in an arbitrary order. However, so far it is not possible to say that process step `D` must be executed after the three process steps `A`, `B`, and `C` have terminated. In order to express this semantics a new modeling element has to be introduced, which is called box.

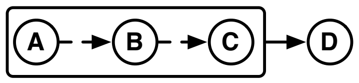

Fig. 5. Box with final acceptance process D

B. The Box Modeling Element

The box modeling element ensures that all the processes inside a box are regarded as a unit. Thus a box can substitute a process. That means that instead of executing a single process A or B the box must be performed, that means the processes within the box must be executed. For instance, in Figure 5 the box must be executed completely before process D can be started. Executing the box means to execute processes A, B, and C in an arbitrary order. This execution results in the following sequences: ABCD, BCAD, CABD, CBAD, ACBD and BACD. D is always the last step that requires the completion of all previous steps respectively the box, in which the steps are contained.

The process of Figure 5 clearly models the medical scenario from Section I, when processes A, B, and C must be performed in any order before process D.

C. Quantification

Often it is necessary to specify that a process can be executed several times. For that purpose we add quantificational aspects to process steps which are novel in the field of process modeling. Every process gets a minimum and maximum counter that indicates how often a process may be executed. If it shall be executed exactly a certain number of times then minimum and maximum are equal. To express that a process step is not essential for the whole process but can be done in the sense of “possible but not necessary”, then a minimum quantification of zero should be selected.

Reconsider the medical example of Section I. We now want to declare how often these processes may be executed:

- Process A: minimum = 1, maximum = 2
- Process B: minimum = 1, maximum = 1 (exactly once)
- Process C: minimum = 0, maximum = * (optional, any repetition)
- Process D: minimum = 1, maximum = 1 (exactly once)

Modeling this scenario by just using conventional, i.e., imperative modeling elements results in a process model as depicted in Figure 6. It is almost impossible to derive the above defined semantics from that diagram. That situation changes when our innovative combination of declarative and imperative modeling constructs is applied. Figure 7 presents the same process model as Figure 6. It is obvious that the complexity of the process model is completely vanished through the use of the new powerful modeling constructs. Users regard processes A, B, and C as a unit that has to be executed before process D can be performed. Also the flexibility of executing the processes within the box can be recognized easily. The number attached to the processes shows how often processes have to be executed. This example nicely shows that the declarative process modeling constructs facilitate compact modeling of complex process-based applications. This modeling style is advantageous for business process modeling in such a way that it is now possible to describe very complex business process models in a more elegant and easier to comprehend way.

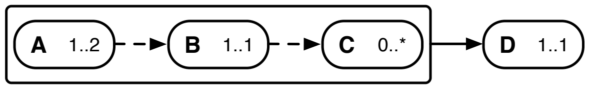

Fig. 7. Simplified medical example

Nevertheless, it is not that easy for novel users to apply our new modeling constructs. Therefore, we were directly supporting them during modeling. We experienced that after a couple of modeling sessions they were able to apply the new constructs independently and, finally, that it was even easier for them to express their complex application scenarios. Without any doubt by introducing our new modeling constructs we are leaving the realm of standards (e.g., BPMN). However, this is not an unusual approach. In many publications (see the BPM conference series [16]) new modeling constructs are introduced, which are not covered by a standard like BPMN. Many researchers and especially practitioners accept that in special cases standards have to be violated in order to provide more adequate modeling capabilities. The tradeoff between standard conformance and enhanced expressiveness has to be resolved individually for each project.

V. Architecture and Implementation

In this section, we want to give an overview on our implementation for process management, i.e., process modeling and execution. In Figure 8 the architecture of our process management infrastructure is depicted.

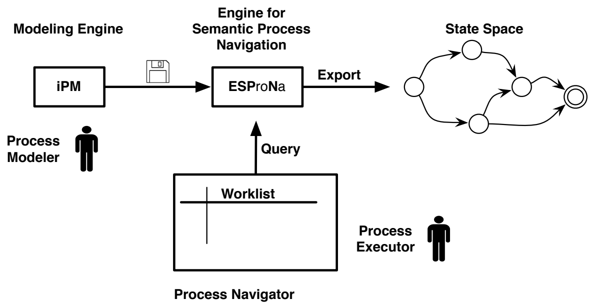

Fig. 8. Architecture of the process modeling and execution framework

Processes are modeled with the tool “iPM Process Modeler” [17]. This tool supports modeling constructs of POPM and all the modeling elements introduced in Section IV. A process model is saved in a special data format and is loaded into the planning component of the process execution system. We call this prototype ”Engine for Semantic Process Navigation” (ESProNa). A process executor (e.g., nurse, medical doctor)

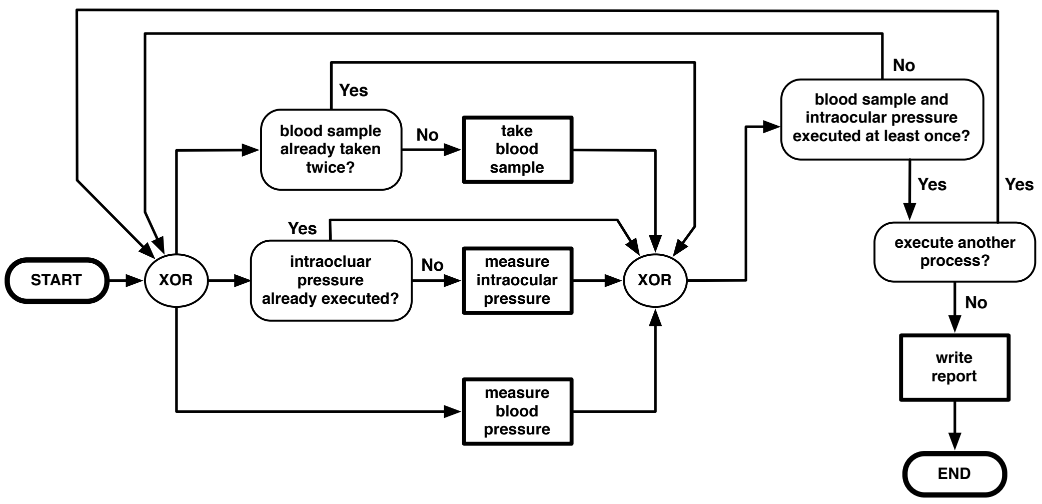
START
XOR
blood sample already taken twice?
Yes
No
take blood sample
intraocluar pressure already executed?
Yes
No
measure intraocular pressure
measure blood pressure
XOR
blood sample and intraocular pressure executed at least once?
No
Yes
Yes
execute another process?
No
write report
END

Fig. 6. Conventional modeling of medical example

can then use the Process Navigator PN [18] to navigate through a process. PN is a process execution system that works on process models deploying the Perspective Oriented Process Modeling method. A work list depicts all processes that are executable. The process executor can then select one of the executable processes to perform it. So far the PN cannot interpret process modeling elements as presented in Section IV. Our prototype ESProNa extends PN in order to cope also with these declarative modeling constructs.

A conventional process management system [19] cannot support a look ahead to the process executor such that he can see what processes are not executable anymore and what processes are still executable. This functionality is additionally provided for PN by ESProNa. With this "look ahead" some kind of guidance is provided to the process executor since he can better anticipate the impact of the execution of a special process. It is of enormous importance when flexible execution is provided as described in Section I and IV. Through the many different alternative execution paths that become available a process executor might get overburdened with the selection of processes for execution. Therefore, this guidance functions is very important and sustains a better overview on process execution. Thus we can say: introducing this highly flexible execution semantics, which drastically reduces path complexity (Section I) comes with costs: loss of overview since often very many processes are executable (what is an indication of flexible execution). However, in our approach this loss is totally compensated through the provided guidance functionality, which will support the process executors to navigate through the process. How to implement guidance? We have chosen a pretty handy approach. Instead of solely offering possible next processes (for execution) we additionally offer two more columns on a work list. These columns depict the following two sorts of processes, which support a look ahead:

- Processes that can still be executed eventually after a now executable process is performed.
- Processes that never can be executed again after a now executable process is performed.

History: -

possible next	some when possible	not possible afterwards
A	A, B, C, D	
B	A, C, D	B
C	A, B, C, D	

History: AB

possible next	some when possible	not possible afterwards
A	C, D	A, B
C	A, C, D	B
D	-	A, B, C

Fig. 9. Work list for the process executor

Figure 9 depicts the implementation of this new kind of worklist. In conventional process execution systems only the left column of the two work lists ("possible next") in Figure 9 would be supported: this column depicts the processes that are executable next. The two columns "some when possible" and "not possible afterwards" depicts processes, which are still executable respectively not anymore executable after a certain process is selected for execution.

Figure 9 shows two different situations whereas the example from Figure 7 is referred to. The upper work list depicts the state when nothing is done yet (history is empty "-"). The three processes of the box are executable (`A, B, C`); `D` is not executable since first the (elements in the) box must be performed. Selecting processes `A` or `C` means that all processes `A`, `B`, `C` and (later) `D` can still be performed again. Selecting process `B` means that `B` must not be performed again since it is only allowed to be performed exactly once (quantification). The lower work list shows a situation where processes `A` and `B` were already performed: `History AB`. Process `B` is not executable any more since it was already executed and the domain constraint (`B` must be executed exactly once) is prohibiting this. Process `D` became executable since the box could be terminated and all processes of the box are performed as often as required minimally. If process `A` is selected it must not be performed again since it can be executed twice at most. When process `C` is selected, then all processes except `B` are executable. In the case that process `D` is selected no other preceding process is executable anymore. It shall be mentioned, that this behavior is exchangeable. We can adopt them to any special business process execution semantics. For example, in the former medical example processes `A`, `B` and `C` are not executable again since `D` has started and the box containing processes `A`, `B` and `C` must not be executed any more.

VI. Conclusion and Outlook

Two observations become noticeable when conventional process execution (PN) is extended with the ESProNa framework: Process modeling can cope with much more complex process models without enhancing complexity, i.e., especially path complexity is well coped with (Section IV). Through the powerful implementation, process executors can effectively be guided through process execution by supporting guidance in form of a "look ahead".

Together, compact process modeling capabilities and powerful process execution guidance provides an add-on to conventional process management that is heavily requested in literature [20], [21], [22].

ESProNa is part of the ForFlow Process Navigator. This system is developed in the joint research project ForFlow [23] among 4 Bavarian Universities and about 30 industrial partners. The Process Navigator is meanwhile in prototype use in 5 partner companies, which intend to use it in productive mode.

The most important next step in our research is to integrate the operational, behavioral and data perspective of process management. We currently investigate how these aspects can be integrated into the ESProNa Framework. First steps into this research are showing that these perspectives can seamlessly be added to the concepts defined so far. However, it is obvious that these perspectives have a major impact on execution flexibility. In a future paper, we will also analyze how workflow patterns [4] can be expressed by ESProNa to show the completeness of our modeling approach.

References

[1] D. Fahland, J. Mendling, H. Reijers, B. Weber, M. Weidlich, and S. Zugal, "Declarative vs. Imperative Process Modeling Languages: The Issue of Maintainability," in *1st International Workshop on Empirical Research in Business Process Management (ER-BPM'09)*, B. Mutschler, R. Wieringa, and J. Recker, Eds., Ulm, Germany, Sep. 2009, pp. 65–76, (LNBIP to appear).

[2] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, and N. Russell, "On the Suitability of BPMN for Business Process Modelling," in *Business Process Management*, 2006, pp. 161–176.

[3] S. White. (2004, May) Business Process Modeling Notation (BPMN) — Version 1.0. [Online]. Available: http://www.bpmn.org/Documents/BPMN_V1-0_May_3_2004.pdf

[4] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[5] OASIS. (2010, Mar) Web services business process execution language version 2.0. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[6] P. Wohed, N. Russell, A. H. M. ter Hofstede, B. Andersson, and W. M. P. van der Aalst, "Patterns-based evaluation of open source BPM systems: The cases of jBPM, OpenWFE, and Enhydra Shark," *Inf. Softw. Technol.*, vol. 51, no. 8, pp. 1187–1216, 2009.

[7] jBPM 3.1 Workflow and BPM made practical, Chapater 9.6 Superstates. IBM. [Online]. Available: http://docs.jboss.com/jbpm/v3.1/userguide/en/html/processmodelling.html

[8] J. Mettraux. (2010, March) ruote 2.1. [Online]. Available: http://ruote.rubyforge.org/

[9] W. M. P. van der Aalst and Ter, "Yawl: yet another workflow language," *Information Systems*, vol. 30, no. 4, pp. 245–275, June 2005. [Online]. Available: http://dx.doi.org/10.1016/j.is.2004.02.002

[10] Z. Zhou, S. Bhiri, and M. Hauswirth, "Control and data dependencies in business processes based on semantic business activities," in *iiWAS'08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*. New York, NY, USA: ACM, 2008, pp. 257–263.

[11] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg, "Declarative workflows: Balancing between flexibility and support." *Computer Science — R&D*, vol. 23, no. 2, pp. 99–113, 2009.

[12] S. Jablonski, "Functional and behavioral aspects of process modeling in workflow management systems," in *CON'94: Proceedings of the Ninth Austrian-informatics conference on Workflow management: challenges, paradigms and products*. Munich, Germany, Germany: R. Oldenbourg Verlag GmbH, 1994, pp. 113–133.

[13] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst, "DECLARE: Full support for loosely-structured processes," in *EDOC'07: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*. Washington, DC, USA: IEEE Computer Society, 2007, p. 287.

[14] S. Goedertier, R. Haesen, and J. Vanthienen, "EM-BrA^{2}CE v0.1: A vocabulary and execution model for declarative business process modeling," K.U.Leuven, FETEW Research Report KBI-0728, 2007.

[15] S. Jablonski and C. Bußler, "Workflow-management: Modeling concepts, architecture and implementation," International Thomson Computer Press, 1996.

[16] U. Dayal, J. Eder, J. Koehler, and H. A. Reijers, Eds., *Business Process Management, 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009. Proceedings*, ser. Lecture Notes in Computer Science, vol. 5701. Springer, 2009.

[17] S. Dornstauder, *Handbook of the iPM Integrated Process Manager*, ProDatO Integration Technology GmbH, 2005.

[18] M. Faerber, S. Jablonski, and S. Meerkamm, "The ProcessNavigator — Flexible process execution for product development projects," *International Conference on Engineering Design, ICED'09*, 2009.

[19] F. Leymann and D. Roller, *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, September 1999.

[20] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke, "A comprehensive approach to flexibility in workflow management systems," in *WACC*. ACM, 1999, pp. 79–88.

[21] R.-M. Stefanie, R. Manfred, and D. Peter, "Correctness criteria for dynamic changes in workflow systems: A survey," *Data & Knowledge Engineering*, vol. 50, no. 1, pp. 9–34, 2004.

[22] W. Aalst and S. Jablonski, "Editorial: Flexible Workflow Technology Driving the Networked Economy," *International Journal of Computer Systems, Science, and Engineering*, vol. 15, no. 5, pp. 265–266, 2000.

[23] Prof. H Meerkamm and Dr.-Ing. K. Paetzhold, *Bayerischer Forschungsverbund für Prozess- und Workflowunterstützung zur Planung und Steuerung der Abläufe in der Produktentwicklung*, ISBN 978-3-9808539-7-2.