Incremental Development of Business Process Models

Carlo Simon
Universität Koblenz-Landau
Universitätsstraße 1, D-56070 Koblenz
simon@uni-koblenz.de

Abstract: The purpose of formal and so called semi-formal approaches to business process modeling is to provide a general technique for the development of workflow based information systems. For this, a concise and repeatable transformation of initial requirements into the final system must be achieved. Moreover, there is also a need for complex models to include justifications that explain the meaning and purpose of each component of the entire model. However, this is not sufficiently supported by current graphical methods. This paper therefore introduces a formal language for process modeling which allows incremental development, graphical visualization of the model, and adding justifications to the model.

1 Introduction

Nowadays, business processes are central for building information systems. Finding and formulating of requirements must be done by computer scientists, business process analysts and domain experts. Especially this last group is typically not familiar with formal, mathematical notations. This motivates the use of more intuitive graphical methods such as event-driven process chains (EPC) by Scheer [HKS93, KNS92, Sc00], Workflow nets by van der Aalst [Aa96, AH02], UML diagrams [RJB99], or the models of the MEMO approach by Frank [Fr02]. Oberweis [Ob96] and Hagen/Stucky [vHS04] also discuss Petri nets as a graphical language for business process modeling. This list is surely not complete.

Oberweis propagates visual models in contrast to pure textual methods, but also points out that the appropriateness of the models' granularity with respect to the addressed reader must be given [Ob96, p. 33-34]. The contradiction that lies between these two requirements is discussed in the next section. As will be shown, even small business processes lead to graphical business process models which are difficult to read due to the natural restriction of humans' reception. Formal textual representations could be helpful for a more precise specification of the required behavior as long as visualizations can also be generated from these specifications.

This paper therefore introduces a formal process language and a novel approach to incrementally develop business process models with the aid of this language. The meaning of the words of this language is defined via their implementation as Module nets which are closely related to Workflow nets. The textual representation of business processes allows to formulate the processes rapidly and to easily augment the specification by justifications for each aspect of the process. Moreover, it allows producing a graphical visualization which is assumed to be easier to understand than the textual form. As a result, this approach does not enforce a contemplator to comprehend the entire model at once. Alternatively, a presentation of the model can incrementally visualize each specified aspect step-by-step together with reasons why this aspect has been included.

The paper is organized as follows: using two EPC business process models taken from the literature, problems concerning their interpretation that result from the absence of justifications are discussed. One of them is then used as a base for the definition of a Workflow net which specifies the behavior precisely. This model, however, is difficult to read because of its complexity. As a consequence, a process specification language is introduced and applied to the example. This offers the opportunity to add justifications and to develop and present the model incrementally explaining each of its aspect hereby. The paper proceeds with an explanation on how to integrate other views on information systems such as information objects or resources into the process specification. It ends with a conclusion.

2 Complexity of Graphical Business Process Models

The first business process model of this paper shown in Figure 1 is taken from Krcmar who describes a decision process with the aid of an EPC [Kr00, p. 115].

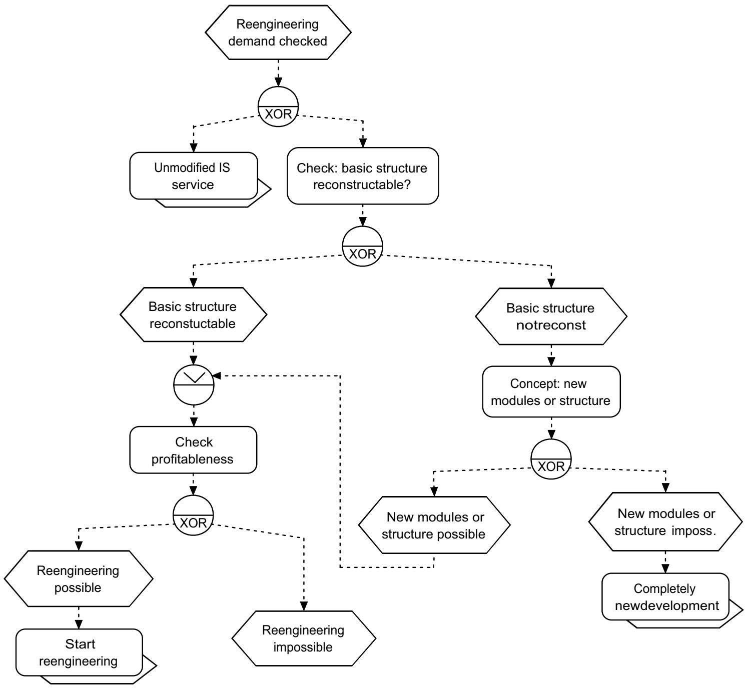
Reengineering demand checked
XOR
Unmodified IS service
Check: basic structure reconstructable?
XOR
Basic structure reconstuctable
Basic structure notreconst
Concept: new modules or structure
XOR
Check profitableness
XOR
New modules or structure possible
New modules or structure imposs.
Reengineering possible
Completely newdevelopment
Start reengineering
Reengineering impossible

Figure 1: Decision on a software reengineering project according to [Kr00, p. 115]

While in its basic structure the EPC has the shape of a binary decision diagram [Ak78] it encloses a submodule for profitability evaluation used under two conditions: if the basic structure can be reconstructed or otherwise new software modules or a new structure can be developed. The integration of the profitability evaluation in different contexts is achieved by an or-connector which precedes the respective part of the model.

Although the model gives a good overview of the decision process, there are some questions left open. What happens if a reconstruction of the basic structure or the development of new modules is not profitable? If the basic structure cannot be reconstructed profitably, is the development of new components probably more reasonable? And if this is impossible, what about the development of a completely new system? Obviously, the model lacks some additional documentation that gives answers to these questions.

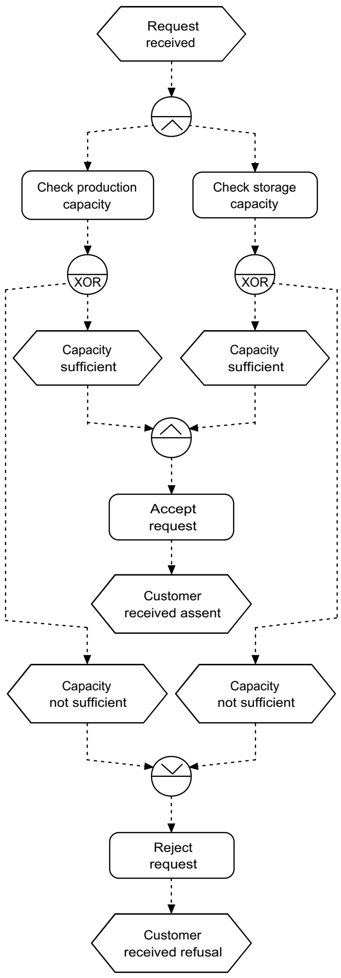

Figure 2: EPC: evaluation customer request

The second business process model is part of a real-world scenario described by Staud [St01, pp. 67-79]. Figure 2 shows the EPC model of, again, a decision situation. In the scenario, a manufacturer produces special machines on customers' requests. For a decision whether a request can be fulfilled or not, both storage and production capacity have to be taken into account. If one of them is insufficient, then the request must be rejected but is accepted otherwise.

Again, the EPC model gives a good overview over the problem, but, does not answer questions concerning the detailed process flow especially in situations where only one resource is insufficiently given (called single failure case): if one type of capacity - for example the storage capacity - is sufficiently given, then this will initiate a reservation of that capacity for the request. If now the other type of capacity is insufficient, then this previous reservation must be released, i.e. there is a need for a rollback on the reservation data. Such a rollback, however, is not represented in the model.

A second aspect under which the model is not yet precise is the situation in which one of the capacities is recognized to be insufficient while a check for the other one has not been conducted yet. Must this check be conducted afterwards or not?

Unfortunately, the situation is not improved if the model is transformed automatically into a model with state semantics as proposed by Kindler [Ki04, CK04], because the absence of a rollback mechanism in the EPC model also prevents the existence of such a mechanism in a model with state semantics. What can be done is to take the EPC as a requirements document for the development of a Workflow net. Figure 3 shows this Workflow net. Within this model, three failure situations for the case where the customer request cannot be fulfilled are distinguished. Moreover, rollback transitions are introduced which support the rollback of storage and production capacity. Finally, the model is sound [Aa96], i.e. each reachable state allows a legal termination of the started process.

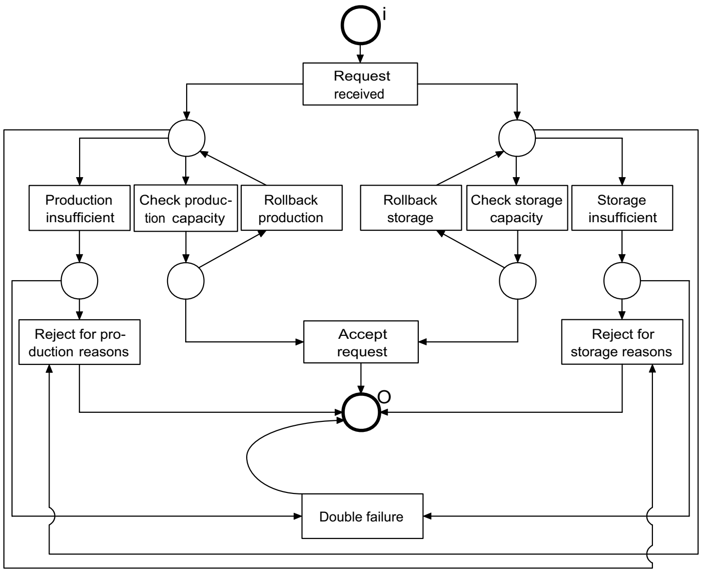

Figure 3: Workflow net with process termination in case of insufficient capacities

The price that has to be paid for precision and soundness is that the model is definitely harder to read than the previous one. For example, the role of the rollback transitions is incomprehensible without the previous explanation, because the model does not contain such reasons. Also the various failure recognition and handling situations appear unmotivated in the final model and surely need to be explained.

3 Modeling Processes with a Formal Textual Language

The simple examples in the previous section motivate a new approach that separates modeling from presentation and supports a step-by-step presentation style which explains the model to a viewer. Especially this last reason is beyond the requirements formulated in the principles of proper modeling (Grundsätze ordnungsgemäßer Modellierung, GoM, [Sc98, pp. 119-137]), because these principles neither cover dynamic aspects of the model development nor its presentation. The requirements formulated by Oberweis [Ob96, p. 34] explicitly mention dynamic aspects of the modeling tasks, but do not include the dynamic presentation of reasons as part of the documentation.

The formal process modeling language introduced here combines textual specification and graphical visualization. The textual formalization goes back onto prior work on a Logic of Actions (LoA) where modules specify sets of sequential processes [Ge73, Ge78, Fi93, LS00, Si01]. In the past, applications of this theory have mainly been in the specification and verification of process behavior as described in [SR04, SD04]. Two extensions of LoA discussed here allow the application of this formal language to business process modeling:

- In LoA, modules - the formulas of the logic - are interpreted by sets of sequential processes which are explicitly defined. The process sets are, however, very complex and are difficult to read. The new approach presented here is based on canonical building rules over modules which take modules as input and generate Module nets out of them. Now, the sequential processes specified by a module are defined as the processes of its Module net implementation. Hereby, processes are specific firing sequences as explained later.
- LoA modules do not support the specification of exceptional behavior in addition to norm behavior. The examples of the previous section, however, have demonstrated the necessity of exceptions for the specification of business processes. Moreover, each of these exceptions is based on reasons which need to be explained while they are added to an existing module.

In the process language introduced here, processes are specified with the aid of modules. Existing modules are probably extended by exceptional behavior. The initial definition of modules is formulated by elementary actions which either occur or are forbidden ($[a]$ or $[\overline{a}]$). With respect to two modules M_1 and M_2, non-elementary modules are built for sequence ($[M_1 \olessthan M_2]$ and $[M_1 \ogreaterthan M_2]$), exclusive alternative and alternative ($[M_1 \oplus M_2]$ and $[M_1 \ovee M_2]$), concurrent behavior which is, however, joined over shared actions ($[M_1 \owedge M_2]$), iteration ($[M_1^*]$, $[M_1^+]$ and $[M_1^n]$), and complement building ($\overline{[M_1]}$). Finally, the coincident occurrence of actions or their prohibition in a single synchronized step is defined with a further operand ($[M_1 \circledast M_2]$).

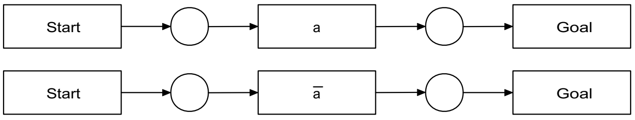

Figure 4: Elementary Module nets over an action a

Modules are implemented by Module nets. The close relationship of these nets with Workflow nets has been shown by Dehnert [De03, p. 82]. Module nets have an explicit *start* transition with empty preset and an explicit *goal* transition with empty postset. Processes of Module nets are all firing sequences which reproduce the empty initial marking by firing *start* and *goal* transition exactly once.

The Module net implementations of elementary processes are predefined as shown in Figure 4. In principle, these implementations consist of only a single transition which represents the occurrence or prohibition of an action a framed by process start and end.

For non-elementary modules, canonical implementation rules take the operands' Module net implementations as input and generate composite Module nets upon them. The building rules are left away here in order to have more space for the demonstration of the application. Now, the semantic of a module is defined as the set of (sequential) processes of its implementation. Figure 5 shows how these concepts are related to each other.

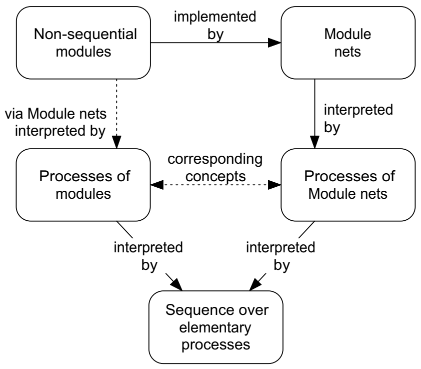

Figure 5: Relationship between the concepts

The norm behavior of the second example of Section 2 can now be defined as follows:

$$\begin{aligned} \mathit{Check} \quad := \quad & \mathit{Request\ received} \olessthan (\\ & \quad \mathit{Check\ production\ capacity} \owedge \mathit{Check\ storage\ capacity} \\ &) \olessthan \mathit{Accept\ request} \end{aligned}$$

Within this module, both *check* actions describe that production or storage capacity are sufficiently given and reserved (in the information system) for the specific customer request. Figure 6 shows the Module net implementation of the norm behavior.

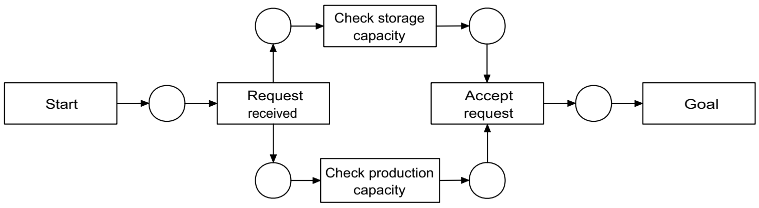

Figure 6: Module net of the norm behavior

Now, this norm behavior must be extended incrementally by exceptional behavior. Such an exception to the norm process

- has to be recognized,
- causes an exceptional behavior, and finally
- has to enable the *goal* transition immediately or alternatively, must produce a marking from which the subsequent firing of the *goal* transition is enabled.

Since a module only specifies processes over elementary actions, the only (process) state information expressible in a module is also related to actions or refers to process start or end. A process state related to an action a can either be $a\bullet$ which indicates the state reached immediately after a has occurred or $\bullet a$ which indicates the state immediately before a can occur. Moreover, *start*$\bullet$ indicates the state at process start and $\bullet$*goal* the state at process end. With these process state related information, a module M is extended by an expression

$$M \cdot\!= \textit{pre} \oslash \textit{Exceptional behavior} \oslash \textit{post}$$

where *pre* is the process state the exception starts from, *Exceptional behavior* is the intended behavior in case of the exception, and *post* is the process state in which norm and exceptional behavior are conjoined.

In the example, the recognition of an insufficient production capacity is specified by

$$\begin{array}{lll} \textit{Check} & \cdot\!= & \bullet\textit{Check production capacity} \oslash (\\ & & \quad \textit{Production insufficient} \oslash \textit{Reject for prod. reasons} \\ & &) \oslash \bullet\textit{goal} \end{array}$$

The related Module net implementation is shown in Figure 7. The figure emphasizes the recognition of the exceptional behavior.

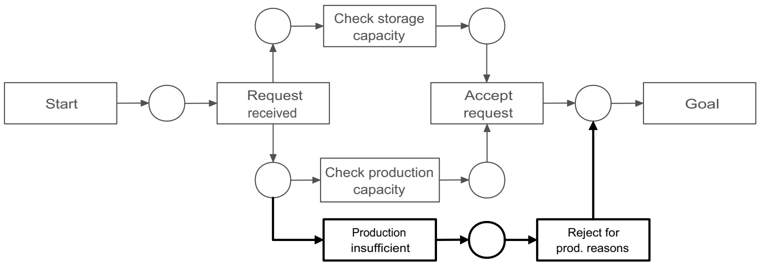

Figure 7: Module net with recognition of insufficient production capacity

Unfortunately, in the Module net implementation of Figure 7 no process exists in which *Reject for prod. reasons* occurs, because whenever the corresponding transition fires one of the places belonging to the check of the storage capacity stays marked and the empty initial marking cannot be reproduced. What is needed in this case is a second kind of extension which alternatively synchronizes concurrent behavior if the synchronization within the norm behavior fails. In general, a module M must be extended by an expression

$$M \cdot\!= pre \oslash Synchronizing\ action$$

where *pre* is a process state concurrently enabled besides an exceptional behavior and *Synchronizing action* is an action of the exception which, by this extension, implements an alternative synchronization.

In the example, this can be achieved by

$$Check \quad \cdot\!= \quad \bullet Check\ storage\ capacity \oslash Reject\ for\ prod.\ reasons$$

Figure 8 shows the implementation of module *Check* so far and emphasizes all aspects of the exceptional behavior in the case of an insufficient production capacity.

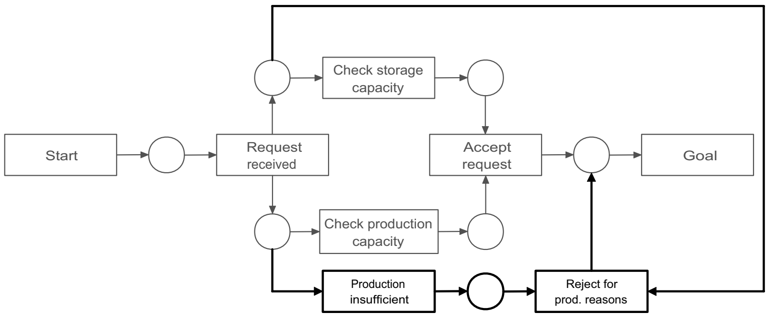

Figure 8: Module net with process termination in case of insufficient production capacity

The recognition and handling of an insufficient storage capacity is now specified analogously by

$$\begin{aligned} Check \quad \cdot\!= \quad & \bullet Check\ storage\ capacity \oslash (\\ & \quad Storage\ insufficient \oslash Reject\ for\ storage\ reasons\\ &) \oslash \bullet goal \end{aligned}$$

$$Check \quad \cdot\!= \quad \bullet Check\ production\ capacity \oslash Reject\ for\ storage\ reasons$$

Finally, the double failure situation can also be described by two extensions

$$Check \quad \cdot\!= \quad \bullet Reject\ for\ prod.\ reasons \oslash (Reject\ double\ failure) \oslash \bullet goal$$

$$Check \quad \cdot\!= \quad \bullet Reject\ for\ storage\ reasons \oslash Reject\ double\ failure$$

Figure 9 shows the implementation of this module so far. In order to document the stepwise model development, the added exceptions are emphasized.

Still firing sequences exist which do not reproduce the empty initial marking. This is the case if one resource - storage or production - is recognized to be insufficient after the other

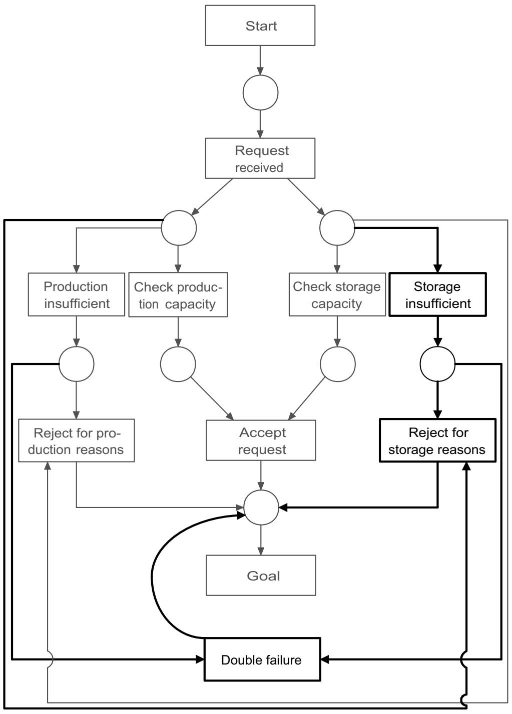

Figure 9: Module net with process termination in case of insufficient capacities

one has already been recognized to be sufficient. Then there must be a rollback on the already reserved resources which is formalized by two extensions

$$\begin{aligned} \mathit{Check} \quad &\cdot\!= \quad \mathit{Check\ production\ capactiy}\bullet\oslash(\\ &\qquad\quad \mathit{Rollback\ production} \\ &\qquad)\oslash\bullet\mathit{Check\ production\ capacity} \end{aligned}$$

$$\begin{aligned} \mathit{Check} \quad &\cdot\!= \quad \mathit{Check\ storage\ capacity}\bullet\oslash(\\ &\qquad\quad \mathit{Rollback\ storage} \\ &\qquad)\oslash\bullet\mathit{Check\ storage\ capacity} \end{aligned}$$

The final Module net is shown in Figure 10. It equals the Workflow net of Figure 3, however, in contrast to the Workflow net it can be presented step-by-step in accordance with its definition. For this, the incremental changes must be emphasized as demonstrated. In an appropriate tool, also the reasons for each extension can simply be explained. This improves the overall comprehension of such a model significantly.

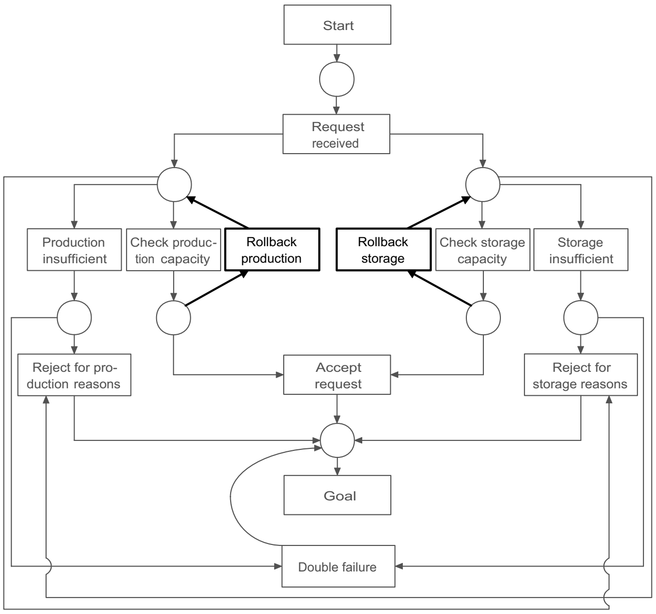

Figure 10: Module net with process termination in case of insufficient capacities

4 View Integration and Semantic

Although applied to business process modeling, the process language introduced in the previous section is an abstract one. The semantic of its words is defined by specific firing sequences of the modules' implementations. This allows to verify the process structures of modules concerning soundness and completeness against given process specifications.

In business process management, the consideration of the pure process aspect is not sufficient. In the architecture of information systems (ARIS), Scheer distinguishes the four perspectives organization, data, function, and control (this is the process view) [Sc00]. Jablonski und Bussler distinguish the perspectives data flow and control flow which are implemented in workflows, workflow applications (i.e. functions), and the organization [JB96, pp. 123-182]. Further perspectives are only shortly discussed. Also Leymann and Roller emphasize process logic, organization and IT infrastructure (the functional and the data view are included into this dimension) as the three dimensions of workflows [LR00, p. 8]. Like these approaches to business process modeling, all others also distinguish their specific views on businesses. They have in common that besides the processes also data, resources, and processing logic need to be represented and, even more important, have to be integrated into each other.

The approach presented here allows such integration by specifying for each action its effect on data objects and other resources, i.e. data flow aspects. The control flow part is completely specified by modules and their Module net implementations. The resulting separation of control and data flow within a single net - however without a formal process specification language - was already discussed by Marx [Ma98].

Data flow aspects of business processes can be added to Workflow nets with the aid of high-level places which might carry structured information [GL81, Ge87, Je92]. Also Vossen points out the important role of high-level nets for business process modeling [Vo05]. As known from database theory, the typical data flow operations are select or test, insert, delete, and update or modify. The implementation of these operations in high-level nets is shown in Figure 11. The data type of a place is specified with the aid of data diagrams or organizational diagrams. Possible bindings of variable X are defined with the aid of transition inscriptions. Finally, a transition is not restricted to access only a single place (i.e. a single type of resource) but might access several resources at once.

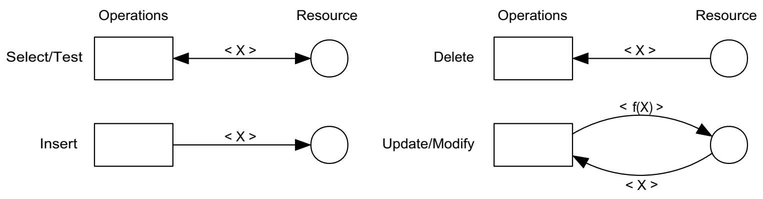

Figure 11: Operations on structured objects with high-level Petri nets

If in addition to the process specification the effect of each action is described with high-level net concepts, a further semantic enrichment of the model is achieved: the (pure) process semantic is extended by a data flow semantic. Consequently, the process language introduced in Section 3 in conjunction with a semantic definition of the actions is called Semantic Process Language (SPL).

5 Conclusion

Using two (small) examples taken from the literature this paper shows three weaknesses of (purely) graphical business process modeling languages:

- Either the models leave room for interpretation, or if they are precise, they reach such a degree of complexity that they are difficult to read and understood by somebody who has not developed the model on her/his own.
- If justifications for each single component of the business process would be added for example with the aid of memos, the readability of graphical models would be further reduced.
- Finally, an incremental presentation of the model together with the arguments for the specific structure is impossible.

Therefore, this paper presents a formal process language as an alternative approach. In contrast to other process languages such as process algebra [Fo00] the graphical visualization of the specified processes is explicitly defined (the problem that process algebra terms cannot be visualized appropriately is widely discussed in the process algebra community, for example in [Ba03, p. 382]). The ability to visualize the processes is achieved with the aid of canonical implementation rules which transform modules - the words of the process language - into Module nets, a variation of Workflow nets. The language supports the definition of norm behavior as well as exceptional behavior. Furthermore, explanations can be added to each definition and extension. These can be presented step-by-step together with each increment. The last section has demonstrated that the approach is open to the integration of data flow aspects.

The semantic process language introduced here also supports an evolution of business processes over time. Gadatsch [Ga03, p. 52] points out the importance of this aspect in his workflow life-cycle model shown in Figure 12.

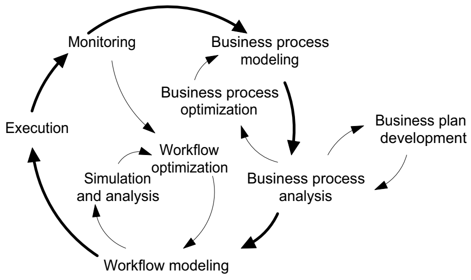
Monitoring
Business process modeling
Business process optimization
Execution
Business plan development
Workflow optimization
Simulation and analysis
Business process analysis
Workflow modeling

Figure 12: Workflow life-cycle model according to [Ga03, p. 52]

If a business process model has to be adapted to a changed reality, each of these changes can be specified and explained as well with the aid of the presented process language. Consequently, the approach introduced in this paper also supports the documentation of business processes throughout their life-cycle.

References

[Aa96] Aalst, W. M. P., van der: Structural Characterizations of Sound Workflow Nets. Computing Science Reports 96/23. Eindhoven University of Technology. 1996.

[AH02] Aalst, W. M. P., van der and Hee, K., van: *Workflow Management - Models, Methods, and Systems*. MIT Press. Cambridge, Massachusetts. 2002.

[Ak78] Akers, S. B.: Binary Decision Diagrams. *IEEE Transactions on Computing*. C(27):509–516. 1978.

[Ba03] Basten, T.: Verifying Petri net Models using Process algebra. In: Girault, C. and Valk, R. (Hrsg.), *Petri nets for Systems Engineering*. chapter 16.5. Springer. Berlin. 2003.

[CK04] Cuntz, N. and Kindler, E.: On the semantics of EPCs: Efficient calculation and simulation. In: Nüttgens, M. and Rump, F. J. (Hrsg.), *EPK 2004: Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, Proceedings*. S. 7–26. Luxembourgh. 2004.

[De03] Dehnert, J.: *A Methodology for Workflow Modeling - From business process modeling towards sound workflow specification*. PhD thesis. TU Berlin. 2003.

[Fi93] Fidelak, M.: *Integritätsbedingungen in Petri-Netzen*. PhD thesis. Universität Koblenz-Landau. 1993.

[Fo00] Fokkink, W.: *Introduction to Process Algebra*. Springer. Berlin. 2000.

[Fr02] Frank, U.: Multi-perspective enterprise modeling (memo) - conceptual framework and modeling languages. In: *35th Annual Hawaii International Conference on System Sciences, HICSS*. S. 75–84. 2002.

[Ga03] Gadatsch, A.: *Grundkurs Geschäftsprozess-Management*. Vieweg. Wiesbaden. 3rd. 2003.

[Ge73] Genrich, H. J.: Formale Eigenschaften des Entscheidens und Handelns. Interner Bericht 09/73-11-29. GMD. St. Augustin. 1973.

[Ge78] Genrich, H. J.: Ein Kalkül des Planes und Handelns. In: *Ansätze zur Organsiationstheorie rechnergestützter Informationssysteme*. GMD Bericht 111. S. 77–92. Oldenbourg Verlag. 1978.

[Ge87] Genrich, H. J.: Predicate/Transition Nets. In: Brauer, W., Reisig, W., and Rozenberg, G. (Hrsg.), *Petri Nets: Central Models and their Properties, Advances in Petri Nets 1986, Part I*. Lecture Notes in Computer Science 254. Springer. 1987.

[GL81] Genrich, H. J. and Lautenbach, K.: System Modelling with High-Level Petri Nets. *Theoretical Computer Science*. 13. 1981.

[HKS93] Hoffmann, W., Kirsch, J., and Scheer, A.-W.: Modellierung mit Ereignisgesteuerten Prozeßketten, Methodenhandbuch. Technical report. Universität des Saarlandes. Institut für Wirtschaftsinformatik, Saarbrücken. 1993.

[JB96] Jablonski, S. and Bussler, C.: *Workflow Management - Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press. London. 1996.

[Je92] Jensen, K.: *Coloured Petri-Nets*. Band 1. Springer Verlag. Berlin. 1992.

[Ki04] Kindler, E.: On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In: Desel, J., Pernici, B., and Weske, M. (Hrsg.), *Business Process Management (BPM 2004)*. volume 3080 of *Lecture Notes in Computer Science (LNCS)*. S. 82–97. Potsdam, Germany. 2004. Springer.

[KNS92] Keller, G., Nüttgens, M., and Scheer, A.-W.: Semantische Prozeßmodellierung auf der Basis Ereignisgesteuerter Prozeßketten (EPK). Technical Report 89. Universität des Saarlandes. Institut für Wirtschaftsinformatik, Saarbrücken. 1992.

[Kr00] Krcmar, H.: *Informationsmanagement*. Springer. Berlin. 2. 2000.

[LR00] Leymann, F. and Roller, D.: *Production Workflow - Concepts and Techniques*. Prentice Hall. Upper Saddle River, NJ. 2000.

[LS00] Lautenbach, K. and Simon, C.: Verification in a Logic of Actions. In: *7. Workshop Algorithmen und Werkzeuge für Petrinetze (AWPN 00)*. Koblenz. 2000.

[Ma98] Marx, T.: *NetCase - Softwareentwurf und Workflow-Modellierung mit Petri-Netzen*. PhD thesis. Universität Koblenz-Landau. 1998.

[Ob96] Oberweis, A.: *Modellierung und Ausführung von Workflows mit Petri-Netzen*. Teubner Studienskripte. Stuttgart. 1996.

[RJB99] Rumbaugh, J., Jacobson, I., and Booch, G.: *The Unfied Modeling Language Reference Manual*. Addison Wesley. Reading, Mass. 1999.

[Sc98] Schütte, R.: *Grundsätze ordnungsmäßiger Referenzmodellierung*. Gabler. Wiesbaden. 1998.

[Sc00] Scheer, A.-W.: *ARIS - Business Process Frameworks*. Springer-Verlag. Berlin. 3rd. 2000.

[SD04] Simon, C. and Dehnert, J.: From Business Process Fragments to Workflow Definitions. In: Feltz, F., Oberweis, A., and Otjacques, B. (Hrsg.), *EMISA 2004 - Informationssysteme im E-Business und E-Government*. Gesellschaft für Informatik, Lecture Notes in Informatics P-56. S. 95–106. Luxemburg. 2004.

[Si01] Simon, C.: *A Logic of Actions and Its Application to the Development of Programmable Controllers*. PhD thesis. Universität Koblenz-Landau. 2001.

[SR04] Simon, C. and Rebstock, M.: Integration of Multi-attributed Negotiations within Business Processes. In: Desel, J., Pernici, B., and Weske, M. (Hrsg.), *Business Process Management (BPM 2004)*. volume 3080 of *Lecture Notes in Computer Science (LNCS)*. S. 148–162. Potsdam, Germany. 2004. Springer.

[St01] Staud, J. L.: *Geschäftsprozessanalyse*. Springer. Berlin. 2. 2001.

[vHS04] von Hagen, C. R. and Stucky, W.: *Business-Process- und Workflow-Management*. Teubner. Stuttgart. 2004.

[Vo05] Vossen, G.: Was Informatiker und Wirtschaftsinformatiker zu Prozessen beitragen. *HMD - Praxis der Wirtschaftsinformatik*. Business Engineering(241):5–6. feb 2005.