Markus Nüttgens, Frank J. Rump,
Andreas Gadatsch (Hrsg.)

EPK 2007

Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten

6. Workshop der Gesellschaft für Informatik e.V. (GI)
und Treffen ihres Arbeitkreises „Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)“

29. November - 30. November 2007 in St. Augustin

Proceedings

Veranstalter

veranstaltet vom GI-Arbeitskreis "Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)" der GI-Fachgruppe WI-MobIS (FB-WI) in Kopperation mit der GI-Fachgruppe EMISA (FB-DBIS) und der GI-Fachgruppe Petrinetze (FB-GInf).

Prof. Dr. Markus Nüttgens (Sprecher)
Universität Hamburg
Email: markus.nuettgens@wiso.uni-hamburg.de

Prof. Dr. Frank J. Rump (stellv. Sprecher)
FH Oldenburg/Ostfriesland/Wilhelmshaven
Email: rump@informatik-emden.de

EPK 2007 / Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. Hrsg.: Markus Nüttgens, Frank J. Rump., Andreas Gadatsch – St. Augustin 2007

Vorwort

Ereignisgesteuerte Prozessketten (EPK) haben sich in der Praxis als Beschreibungsmittel für betriebliche Abläufe etabliert. Mit dem Aufbau der Arbeitsgruppe "Formalisierung und Analyse Ereignisgesteuerter Prozessketten (EPK)" im Jahre 1997 wurde ein erster Schritt unternommen, einen organisatorischen Rahmen für Interessenten und Autoren wesentlicher Forschungsarbeiten zu schaffen und regelmäßige Arbeitstreffen durchzuführen (Organisatoren: Markus Nüttgens, Andreas Oberweis, Frank J. Rump). Im Jahr 2002 wurden die Arbeiten der "informellen“ Arbeitsgruppe in den GI-Arbeitskreis "Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)" der GI-Fachgruppe WI-MobIS (FB-WI) überführt und inhaltlich erweitert. Der 1. Workshop „EPK 2002“ fand im November 2002 in Trier statt, der 2. Workshop „.EPK 2003“ im Oktober 2003 in Bamberg im Vorfeld der 11. Fachtagung „MobIS 2003“, der 3. Workshop „EPK 2004“ in Luxemburg im Rahmen der GI-Fachtagung „EMISA 2004“, der 4. Workshop „EPK 2005“ in Hamburg und der 5. Workshop „EPK 2006“ in Wien statt.

Der Arbeitskreis soll Praktikern und Wissenschaftlern als Forum zur Kontaktaufnahme, zur Diskussion und zum Informationsaustausch dienen. Die Aktivitäten des Arbeitskreises werden unter der Internetadresse http://www.epk-community.de dokumentiert (aktuell: 250 Mitglieder).

Der vorliegende Tagungsband enthält sechs vom Programmkomitee ausgewählte und auf dem Workshop präsentierte Beiträge. Jeder Beitrag wurde einfach-blind und dreifach begutachtet.

Die Beiträge decken ein breites Spektrum zur Spezifikation und Anwendung Ereignis-gesteuerter Prozessketten (EPK) ab:

- EPK Konzepte
- EPKs und Werkzeuge
- EPKs und andere Formalismen
- Qualitätsaspekte von EPKs

Der Tagungsband wurde ausschließlich in digitaler Form publiziert und ist im Internet frei verfügbar (CEUR Workshop Proceedings).

Wir danken der Fachhochschule Bonn-Rhein-Sieg und Herrn Jens Juszczak für die Bereitstellung der Räumlichkeiten und den Autorinnen und Autoren und den Mitgliedern des Programmkomitees für die Beiträge zur Realisierung des Workshops.

Hamburg, Emden und St. Augustin im November 2007

Markus Nüttgens
Frank J. Rump
Andreas Gadatsch

Programmkomitee

Thomas Allweyer, FH Kaiserslautern
Jörg Becker, Uni Münster, ERCIS
Jan vom Brocke, Hochschule Liechtenstein
Jörg Desel, KU Eichstätt
Andreas Gadatsch, FH Bonn-Rhein-Sieg
Ekkart Kindler, DTU Kopenhagen
Peter Loos, Uni Saarland, IWi im DFKI
Jan Mendling, Queensland University of Technology (Brisbane)
Markus Nüttgens, Uni Hamburg (Co-Chair)
Andreas Oberweis, TU Karlsruhe
Michael Rebstock, FH Darmstadt
Peter Rittgen, Uni Boras
Michael Rosemann. Queensland University of Technology (Brisbane)
Frank Rump, FH Oldenburg/Ostfriesland/Wilhelmshaven (Co-Chair)
Carlo Simon, Provadis School of Int. Management and Technology AG, Fft. a. M.
Oliver Thomas, Uni Saarland, IWi im DFKI

Organisation

Andreas Gadatsch, FH Bonn-Rhein-Sieg
Jens Juszczazk, FH Bonn-Rhein-Sieg

Inhaltsverzeichnis

From Process Models to Business Landscapes

Oliver Kopp, Hanna Eberle, Tobias Unger, Frank Leymann
University of Stuttgart, Institute of Architecture of Application Systems
{kopp, eberle, unger, leymann}@iaas.uni-stuttgart.de

Abstract: Today, architecture and business processes are modeled separately. The only integration in architectural diagrams is done with Petri nets in the Fundamental Modeling Concept. Since business users prefer EPCs over Petri nets, we show how information of extended EPCs can be transformed into business landscapes. This facilitates development of IT landscapes satisfying the requirements of the business process and adoption of existing IT infrastructures to new requirements.

1 Introduction and Motivation

Business process models show how business functions relate to each other in a concrete scenario. Business landscapes provide a global view on all business functions and business items within a company. So far, business landscapes and business processes have been modeled separately. The automatic inclusion of the information of business processes in business landscapes has been done manually. In this paper, we present an approach to map the content of business processes to business landscapes. This enables a top-down approach for modeling IT infrastructures: First, the business process is modeled. Second, the presented mapping is used to automatically derive a business landscape. Afterwards, this landscape serves as basis for creating an IT landscape supporting the modeled business process.

Another application scenario is to identify deficiencies in an existing IT infrastructure with respect to a specific business model as illustrated in Figure 1. Using our approach, a business landscape is generated. The elements of this landscape can then be placed in an existing (pure) IT landscape. The result is a combined landscape, which contains both IT and business aspects. The result of the placement helps to identify lacks in the existing infrastructure. For example, the fact that some business items cannot be placed is an indicator that the IT architecture does not fully support the business process.

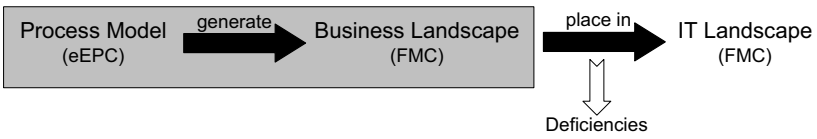


Figure 1: Approach overview

As notation for business processes, we use extended Event-driven Process Chains, which are briefly explained in section 2. We chose the Fundamental Modeling Concepts as notation for business landscapes and present the basics in section 3. The main section 4 is devoted to the mapping of extended Event-driven Process Chains to business landscapes, covering in detail how functions, organizational units and data are mapped.

2 Event-driven Process Chains

We use extended Event-driven Process Chains (eEPCs) as the input for the mapping. eEPCs are Event-driven Process Chains (EPCs, [KNS92]), where functions can be annotated with elements (cf. [STA05], where the usage of eEPCs is illustrated). For our mapping, we restrict the elements to "organizational unit" and "information item." An organizational unit may be connected with the "is involved with" relation. This relation combines all relations of organizational units with functions, such as "executes," "is responsible for" or "has decision making power." An information item represents data, such as "output data." Finally, data can be sent or received by a function. Definition 1 shows a definition of the used variant of eEPCs based on the formalization of [Kin06]. It is important to note that the presented variant of eEPCs is a non-hierarchical EPC.

Definition 1 (Extended Event-driven Process Chain (eEPC))
An extended Event-driven Process Chain (eEPC) consists of events E, functions F, connectors C, control flow arcs $A \subseteq (E \cup F \cup C) \times (E \cup F \cup C)$, a set of names N, a labeling function l, organizational units O, information items I, an involved-with relation $i \subseteq F \times O$, a send relation $s \subseteq F \times I$ and a receive relation $r \subseteq F \times I$. The sets E, F, C, A, N, O and I are pair-wise disjoint.

The labeling function l assigns a name to an element of an EPC: $l : E \cup F \cup C \cup O \cup I \rightarrow N \cup \{and, or, xor\}$. A connector $c \in C$ may only have and, or, or xor assigned. Events, functions, organizational units and information items may only have an element out of N assigned.

$$M = (E, F, C, A, N, l, O, I, i, s, r)$$

To illustrate our mapping, a simplified online shop process is used. A customer places an order at an online shop, which processes the order. In parallel to the goods issue, the invoice is created and delivered. As soon as a payment of the customer is received, the payment is processed and the goods are delivered. Figure 2 presents the eEPC for the online shop.

3 The Fundamental Modeling Concept

The Fundamental Modeling Concept (FMC, [KW03, Tab05]) is a graphical modeling notation, which can be used to describe software systems on both the business and the technological level. FMC allows to visualize complex software systems as a system

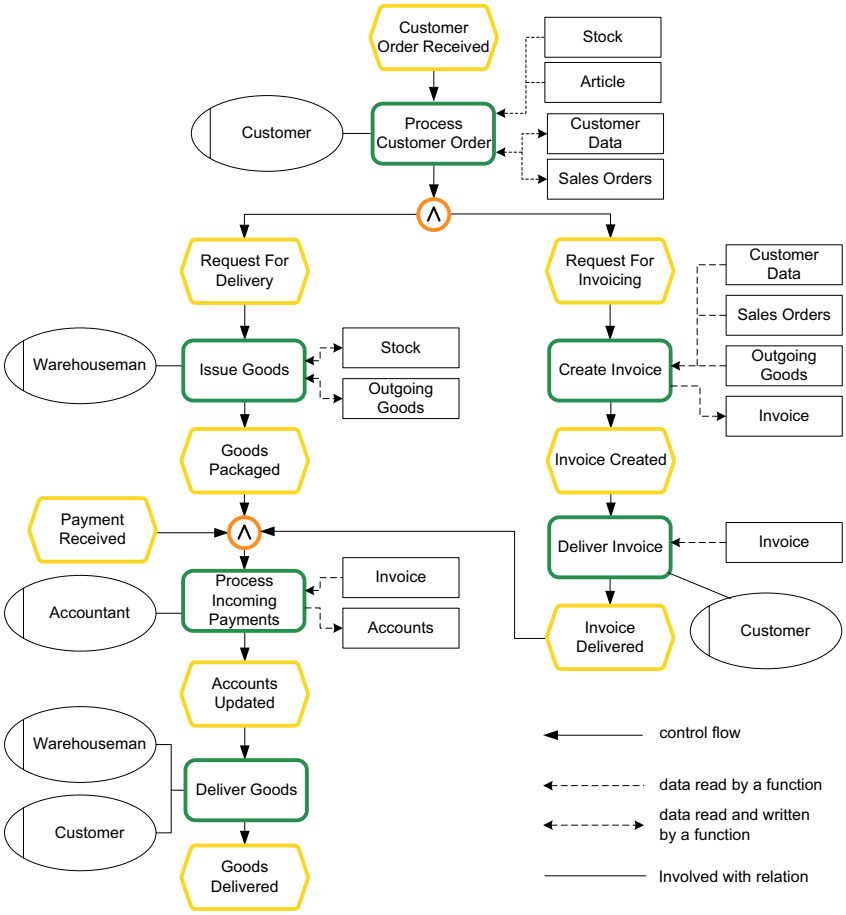


Figure 2: EPC for the example online shop

landscape which provides an overview on the used software, hardware and their relation to each other. FMC was introduced as an intuitive architectural description language to document and communicate a system architecture. It is designed to be simple and universal. Unlike UML [Obj07], FMC describes the structure of a system in a way that is independent of any implementation paradigm and that is on a higher level [KW03].

FMC distinguishes between three structural types: the data diagram, the process diagram and the block diagram. The data diagram is used to capture the relational data model and is similar to entity relationship diagrams. The process diagram expresses process structures using Petri nets. The block diagram is used to capture the components and their relationships. Since both FMC data diagrams and Petri nets are not used by business users, we focus on the block diagrams.

3.1 FMC Block Diagrams

FMC block diagrams are also called landscape architectures and provide an overview of the composite structure of systems [AK07]. In the following, we present all details of FMC block diagrams and present a formalization. The formalization itself does not include the rendering of FMC diagrams and excludes details which are not necessarily needed in the transformation.

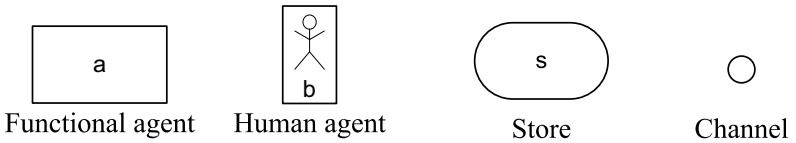
FMC block diagrams consist of components and their relationships. A component can be either active or passive. AC denotes the set of active components. An active component is capable of performing functionality. Active components are divided into functional agents and human agents. Functional agents represent functions with a defined mapping between input and output data. Human agents represent human actors. The function $\mathsf{t_{AC}} : \mathsf{AC} \rightarrow \{\mathit{functional}, \mathit{human}\}$ returns the type of an active component. The function $\mathsf{n_{AC}} : \mathsf{AC} \rightarrow \mathcal{N}$ returns the name of an active component, where $\mathcal{N}$ denotes the set of all names. An active component reads, writes or both reads and writes data.

PC denotes the set of passive components. A passive component is a storage or a channel. Note that passive components are also called "locations" to underline that data is located there. A storage is used by active components to store data and a channel is used for communication purposes between at least two active system components. Storages store data durably. A storage may contain triggers which trigger readers of the storage when a data item has been changed. Triggers do not have a visual representation and are therefore excluded from the formalization.

In contrast to storages, channels do not store data durably. Thus, channels can be regarded as telephone wires: The data is available at one point in time and will be lost afterwards. Hence, it is important that some active component reads it while it is there. The function $\mathsf{t_{PC}} : \mathsf{PC} \rightarrow \{\mathit{storage}, \mathit{channel}\}$ returns the type of a passive component. Thus the implicit sets of storages $\mathsf{PC_S}$ and channels $\mathsf{PC_C}$ are defined:

$$\mathsf{PC_S} = \{s \mid s \in \mathsf{PC},\ \mathsf{t_{PC}}(s) = \mathit{storage}\}$$
$$\mathsf{PC_C} = \{c \mid c \in \mathsf{PC},\ \mathsf{t_{PC}}(s) = \mathit{channel}\}$$

The function $\mathsf{n_{PC}} : \mathsf{PC} \rightarrow \mathcal{N} \cup \{\epsilon\}$ returns the name of a passive component. Storages must take a name and channels may be unnamed (denoted by ϵ). Note that the set $\mathcal{N}$ may not contain ϵ. The graphical notation of FMC components is presented in figure 3.



(a) Active Components (b) Passive Components

Figure 3: FMC Components

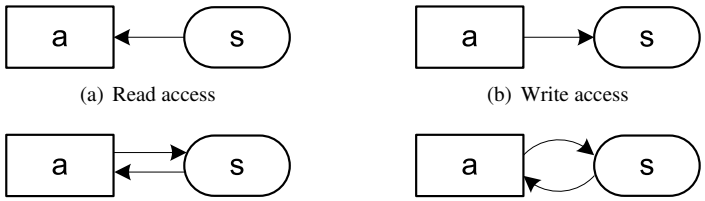
Active and passive components may be connected via connectors denoting their relationship. A relationship is read, write and modify. Read and write are both used at channels and storages. "Modify" may be used at storages to denote that an item in the storage is modified. FMC allows to model channels used for request-response. Therefore "rri" and "rrp" are introduced for channels. The initiator is connected with "rri" to the channel and the replying agent with "rrp". Note that it is allowed to connect an agent to storage with both read and write relations. This states that the agent reads data and writes possibly different data. The set R contains different kinds of relationships: $\mathsf{R} = \{\mathsf{r}, \mathsf{w}, \mathsf{mod}, \mathsf{rri}, \mathsf{rrp}\}$. The set E is the set of all connectors.

$$\begin{aligned}\mathsf{E} \subseteq\ & \mathsf{AC} \times \{\mathsf{r}, \mathsf{w}, \mathsf{mod}\} \times \mathsf{PC_S} \cup \\ & \mathsf{AC} \times \{\mathsf{r}, \mathsf{w}, \mathsf{rri}, \mathsf{rrp}\} \times \mathsf{PC_C}\end{aligned}$$

Since a channel can be either used as request-response channel, as unidirectional channel or bidirectional channel, following property has to be satisfied by E:

$$\begin{aligned}&\forall (a, r, c) \in (\mathsf{E} \cap \mathsf{AC} \times \mathsf{R} \times \mathsf{PC_C}) : \\ &r \in \{\mathsf{r}, \mathsf{w}\} \ \Rightarrow \nexists\, (a, r', c) \in \mathsf{E} : r \in \{\mathsf{rri}, \mathsf{rrp}\} \wedge \\ &r \in \{\mathsf{rri}, \mathsf{rrp}\} \ \Rightarrow \nexists\, (a, r', c) \in \mathsf{E} : r \in \{\mathsf{r}, \mathsf{w}\} \wedge \\ &r = \mathsf{rri} \ \Rightarrow \nexists\, (a, r', c) \in \mathsf{E} : r = \mathsf{rrp} \wedge \\ &r = \mathsf{rrp} \ \Rightarrow \nexists\, (a, r', c) \in \mathsf{E} : r = \mathsf{rri}\end{aligned}$$

Figure 4 presents the graphical representation of connectors between active components and storages and Figure 5 presents the graphical representation of connectors between active components and channels.



(a) Read access (b) Write access

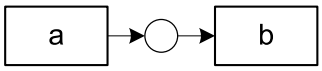
(c) Read and write access (d) Modifying access

Figure 4: Relationships between active components and storages

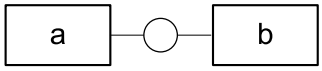
In short, a FMC block diagram is bipartite graph G, where the sets AC and PC form the nodes and E forms the edges between the nodes.

$$G = (\mathsf{AC}, \mathsf{PC}, \mathsf{E}, \mathsf{RR}, \mathcal{N}, \mathsf{t_{AC}}, \mathsf{t_{PC}}, \mathsf{n_{AC}}, \mathsf{n_{PC}})$$

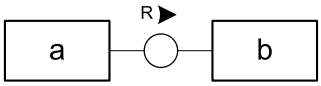
.



(a) Unidirectional channel. Active component a has write access to the channel, whereas active component b has read access to the channel



(b) Bidirectional channel. Both active components a and b have read and write access to the channel



(c) Request-response channel. a is the initiator and b replies to the request.

Figure 5: Relations between active components and channels

3.2 Business Landscapes

FMC can be used to model the architecture from a business and from an IT point of view on any granularity level. Therefore, it is important to state what aspects should be captured by the block diagram. Our work focuses on the business aspect. So called "business landscapes" are used in the requirements engineering and are usually designed by business people [AK07]. Business landscapes do not depict technical information such as servers, clients or databases. The purpose of business landscapes is to visualize

- business functionalities,
- business data,
- business roles and
- their relationships with each other.

Human agents represent business roles, such as members of the sales department. *Functional agents* represent business functionality. *Storages* can either represent stored master data or dynamic data. *Channels* connect agents with each other. The semantics of a connection established via a channel is that information is flowing between the two connected agents. If a storage is connected with two agents, it is not necessary to additionally connect these two agents with a channel.

Figure 6 presents an example business map. It shows the business perspective on the architecture of an online shop.

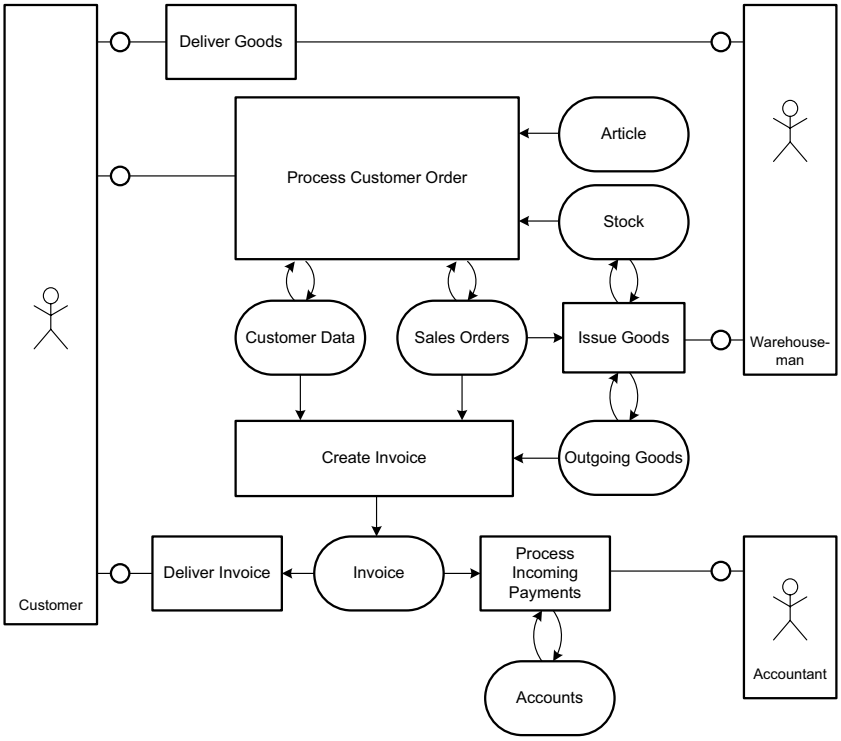


Figure 6: FMC business landscape of an online shop, adopted from [AK07]

3.3 Pure IT Landscapes

Pure IT landscapes are FMC block diagrams capturing technological aspects. They show software components and hardware components and their relationships. For example, a software component is a database and a hardware component is a Unix server. Pure IT landscapes can be used to define new systems and to document already existing ones. New systems can be designed by using architectural patterns, depending on what the requirements on the system are. If existing systems are documented, the resulting landscape can be used to identify bottlenecks and to determine changes needed for improvement of the system. Figure 7 presents the pure IT landscape of the example online shop.

3.4 Relating Pure IT Landscapes with Business Landscapes

Business functionality and business data need an implementation. Therefore, business and pure IT landscapes need to be related. These relations are shown in a mixed landscape, called “IT landscape” [AK07]. IT landscapes are pure IT landscapes, where business functions and data are embodied in the technical components. This approach enables

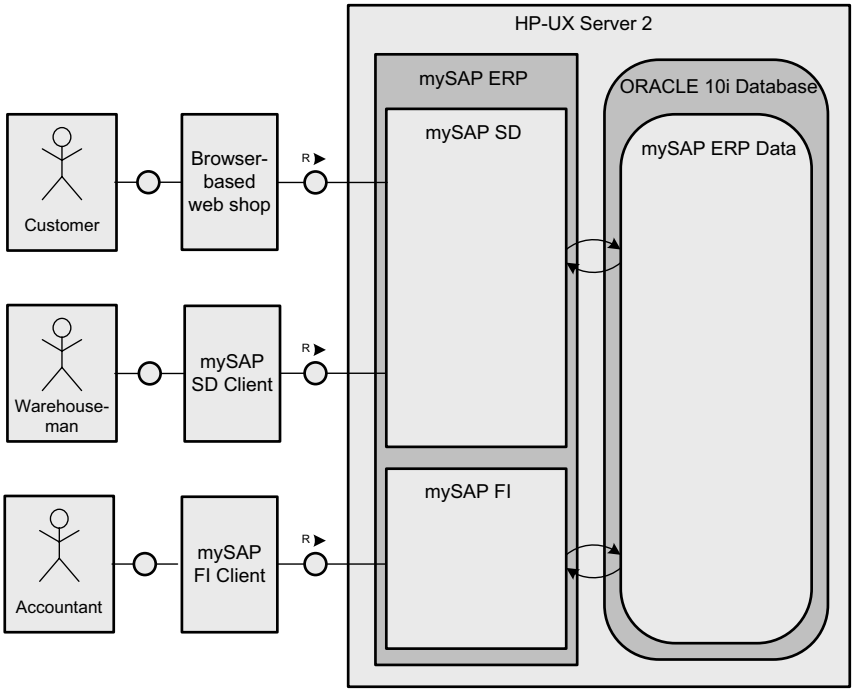


Figure 7: FMC pure IT landscape of an online shop

business analysts and IT professionals to start modeling independently of each other. Figure 8 presents the combined IT and business landscapes. In our application scenario, the IT landscape is the place, where the artifacts of the generated business landscape are placed in to uncover deficiencies of the existing pure IT landscape.

4 Mapping eEPCs to an FMC Block Diagram

Having defined the notation of eEPCs and FMC, we present how to map information from eEPCs to an FMC block diagram. We explain each step, show an algorithm and apply each step to the example online shop to illustrate the mapping. The algorithm maps an eEPC $M = (E, F, C, A, N, l, O, I, i, s, r)$ to a FMC block diagram $G = (\mathsf{AC}, \mathsf{PC}, \mathsf{ERR}, \mathcal{N}, \mathsf{t}_{\mathsf{AC}}, \mathsf{n}_{\mathsf{AC}}, \mathsf{n}_{\mathsf{PC}})$. We assume, that all sets in G are set to empty sets and all functions are undefined.

In FMC, organizational units are seen as business roles, which are represented in the FMC as human actors. The aim of the modeled EPC is to provide a business view on an executable workflow. This implies that each function of the EPC should be executed by a machine. Since a machine is represented as functional agent in FMC, we map functions to functional agents. An organizational unit is connected with a function by the relation “is

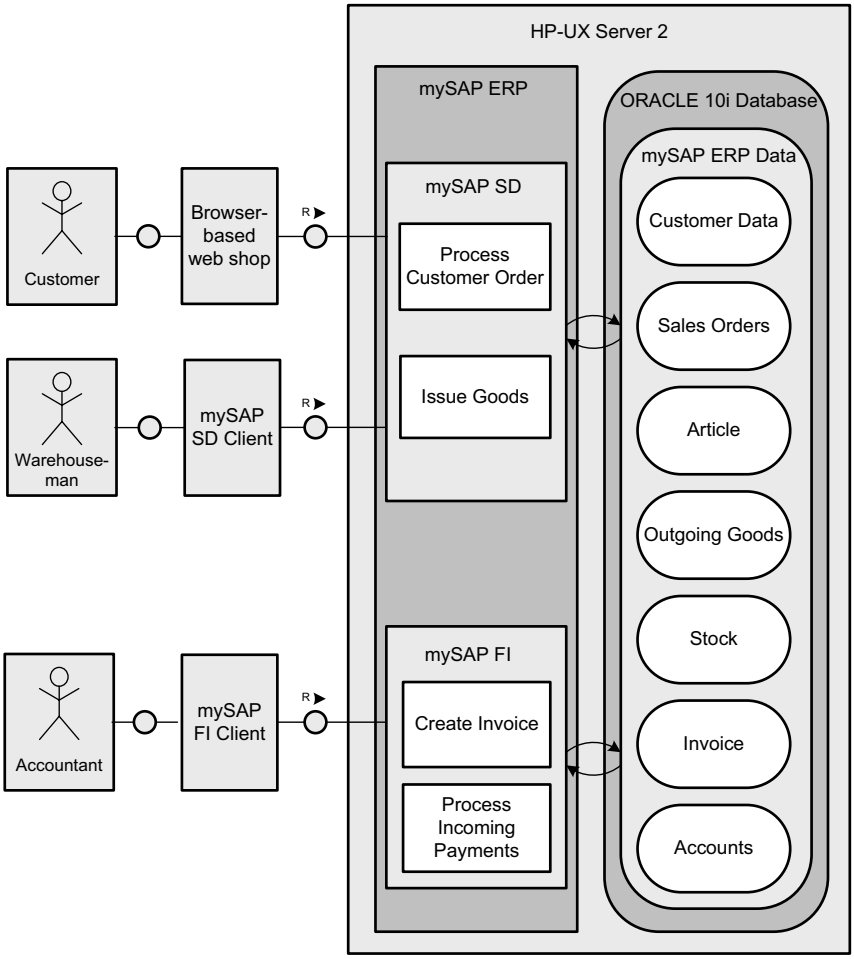


Figure 8: FMC IT landscape of an online shop [AK07]

involved with". That relation does not state any direction of the communication. Therefore, the relation between the human agents and functional agents is realized with a bidirectional channel. A bidirectional channel denotes that the agents are communicating with each other. The mapping of a function to a functional agent can also be interpreted, that the functional agent in the FMC block diagram is "supporting" to human agent to do his task. For example, a tool or a screen form support a human to perform his task.

It is important to note that the names of the artifacts are retained during the mapping. Figure 9 presents the mapping of functions, the involved organizational units and their relation to agents and channels connecting them. Algorithm 1 presents the algorithm. In the algorithm, we used the fact that a function $f : A \rightarrow B$ can also be represented as a set of tuples $f \subseteq A \times B$.

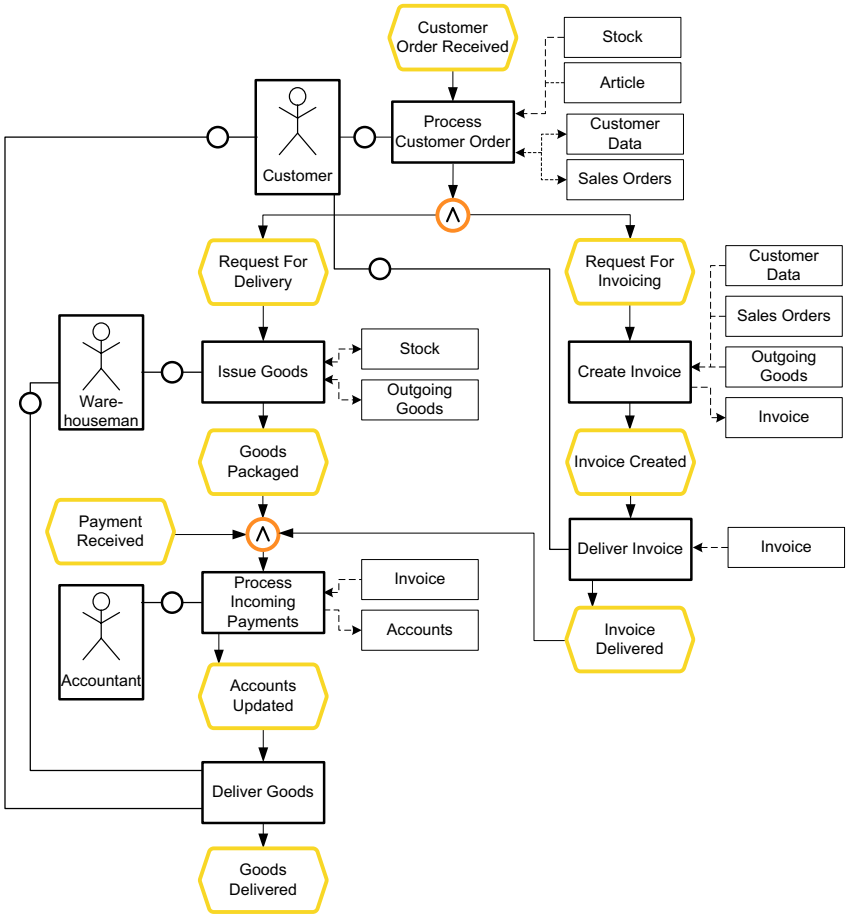


Figure 9: Functions mapped to functional agents. "is involved with" mapped to bi-directional channels.

An information item represents data. Since data is stored in storages in FMC, information items are mapped to storages. If a function receives an information item, it has to read the information item somewhere. Therefore, the read relation is mapped to a connection with the relationship "read". By analogy, the write relation is mapped to a connection with the relationship "write". If a functions both reads and writes an information item, both are summed up in the modifying access relationship. Figure 10 illustrates the mapping using the data accessed by the "process customer order" function as an example. The algorithm is presented in algorithm 2.

So far, the flow relation has not been mapped. Depending on the underlying IT infrastructure, explicit channels are needed to denote that a functional agent triggers a subsequent agent. Therefore, the connection between two functions (with no other function in between) can be mapped to a unidirectional channel. On the other hand, storages may contain triggers.

Algorithm 1 Mapping organizational units, functions and their relation to FMC. Input is an eEPC M, the output is a FMC block diagram G.

procedure CREATEACTORS(M,G)
  $\mathsf{AC} \leftarrow F \cup O$
  $\mathsf{t}_{\mathsf{AC}} \leftarrow \{(ac, functional) \mid ac \in F\} \cup \{(ac, human) \mid ac \in O\}$
  $\mathcal{N} \leftarrow \{n \mid n \in l(e),\ e \in F \cup O\}$
  $\mathsf{n}_{\mathsf{AC}} \leftarrow \{(ac, l(ac)) \mid ac \in \mathsf{AC}\}$
  for all $(f, o) \in i$ **do**
    Create a new channel c
    $\mathsf{PC} \leftarrow \mathsf{PC} \cup c$
    $\mathsf{t}_{\mathsf{PC}} \leftarrow \mathsf{t}_{\mathsf{PC}} \cup (c, channel)$
    $\mathsf{n}_{\mathsf{PC}} \leftarrow \mathsf{n}_{\mathsf{PC}} \cup (c, \epsilon)$
    $\mathsf{E} \leftarrow \mathsf{E} \cup (f, \mathsf{r}, c) \cup (f, \mathsf{w}, c)$
    $\mathsf{E} \leftarrow \mathsf{E} \cup (o, \mathsf{r}, c) \cup (o, \mathsf{w}, c)$
  end for
end procedure

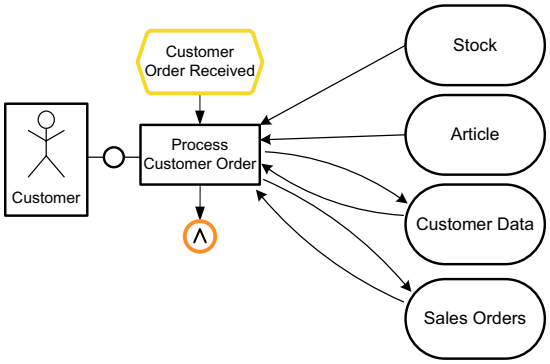


Figure 10: Information items mapped to storages

Algorithm 2 Mapping organizational units, functions and their relation to FMC. Input is an eEPC M and an FMC block diagram G. The block diagram G is modified and thus also an output.

procedure CREATESTORAGES(M,G)
  $\mathsf{PC} \leftarrow \mathsf{PC} \cup I$
  $\mathsf{t_{PC}} \leftarrow \mathsf{t_{PC}} \cup \{(i, storage) \mid i \in I\}$
  $\mathcal{N} \leftarrow \mathcal{N} \cup \{n \mid n \in l(i),\ i \in I\}$
  $\mathsf{n_{PC}} \leftarrow \mathsf{n_{PC}} \cup \{(i, l(i)) \mid i \in I\}$
  $RW \leftarrow s \cap r$ // Function sending and receiving
  for all $(f, i) \in RW$ **do**
    $\mathsf{E} \leftarrow \mathsf{E} \cup (f, \mathsf{mod}, i)$
  end for
  $W \leftarrow s \setminus RW$ // Function only sending
  for all $(f, i) \in W$ **do**
    $\mathsf{E} \leftarrow \mathsf{E} \cup (f, \mathsf{w}, i)$
  end for
  $R \leftarrow r \setminus RW$ // Function only receiving
  for all $(f, i) \in R$ **do**
    $\mathsf{E} \leftarrow \mathsf{E} \cup (f, \mathsf{r}, i)$
  end for
end procedure

For example, an inserted information item can trigger the subsequent agent. Hence, an additional channel is unnecessary in this case. Since triggers are currently not included in the metamodel of FMC, our mapping is not aware of triggers. Furthermore, the Business Process Execution Language (BPEL) is used to drive the business process in a web services architecture [WCL+05]. Thus, the services (modeled by functional agents) do not have to be aware of other services in the business process. Thus, we do not map the connections between functions to channels. As a result, events between functions are not mapped either. There are two mapping options for events without a preceding or subsequent function: (i) induce a human actor and connect it with the actor representing the preceding/subsequent function or (ii) do not map it to a FMC representation. Option (i) is not consistent with the exclusion of channels between functional agents in the mapping. Therefore, option (ii) is taken, which leads to the FMC business landscape presented in figure 6.

Algorithm 3 presents the complete algorithm, which first initializes the target FMC block diagram and calls CREATEACTORS and CREATESTORAGES.

5 Related Work

There exists work on the transformation of graphically modeled business processes to executable workflows (e.g., [ODtHvdA07, MLZ06]). These works do not deal with architecture generation, but can be extended with our contribution. [Asc07] presents how a

Algorithm 3 Mapping an eEPC to a FMC block diagram. Input is an eEPC M, the output is a FMC block diagram G.

procedure MAP(M,G)
  $\mathsf{AC}, \mathsf{PC}, \mathsf{RR}, \mathcal{N}, \mathsf{t}_{\mathsf{AC}}, \mathsf{n}_{\mathsf{AC}}, \mathsf{n}_{\mathsf{PC}} \leftarrow \emptyset$
  CREATEACTORS(M, G)
  CREATESTORAGES(M, G)
end procedure

FMC business landscape can be manually created out of an EPC. The generated business landscape contains more information than the EPC alone. For example, actors performing functionality in the same area, are nested in an actor representing functionality in that area. Our work extends that work by automating the mapping of the EPC elements to FMC business landscapes, which is a basis for the subsequent step of structuring and simplifying the business landscape.

[Mol05] describes a process-oriented application landscape with IT modeling as one aspect. The work states that an IT model cannot be automatically generated out of a business process due to different modeling domains. Instead of automatic generation, required control structure and data can be mapped to generic methods and objects. Our work supports this mapping by automatically generating required artifacts. These artifacts can then be placed in the existing IT landscape.

The Rational Unified Process [Kru04] is an engineering process, which describes phases for software development. Processes can be modeled as EPCs in the "business modeling discipline". In the following discipline "Requirements discipline", the generated business landscapes can be used as communication vehicle between the developers and the customers.

6 Conclusion and Outlook

We presented the concept of mapping extended Event-driven Process chains (eEPCs) to FMC business landscapes. This enables a top-down approach for modeling IT infrastructures: First, the eEPC for the business process is modeled. Afterward, the presented mapping is used to derive a FMC business landscape. This landscape can then be used as basis for creating an IT landscape supporting the modeled business process. The approach can also be used to identify the gaps between the existing IT infrastructure and the infrastructure needed for process execution.

Another application scenario is to map multiple eEPCs to a single business landscape. This business landscape captures the requirements of all given processes at one place. Since the generated business landscape provides another view on the modeled processes, the landscape can be used as additional validation of the modeled processes by the business process designers.

The generated business landscape is consistent with the business process defined in eEPC. That means, the required business functions, the required storages and the participating

human actors are – by construction – all contained in the business landscape. Thus, defining or checking the IT using the generated business landscape ensures that all required business functionality is covered by the IT.

Ongoing work is the implementation of the approach in Arcway Cockpit [AG07] and a customer evaluation. Future work is the support of eEPCs containing information about the IT infrastructure. For example, a machine can execute a function, which results in a nesting of the generated functional agents. We plan to define FMC4SOA, which includes special markers for an SOA IT infrastructure at FMC block diagrams. This allows for using EPCs with technical details (e.g., [ZM05]) can be used as input of our approach. The technical details are then stored in an FMC4SOA landscape.

Acknowledgments

Oliver Kopp is funded by the German Federal Ministry of Education and Research (project Tools4BPEL, project number 01ISE08).

References

[AG07] Arcway AG. Homepage of Arcway Cockpit, 2007. `http://www.arcway.com`, visited 2007-09-15.

[AK07] P. Aschenbrenner and F. Keller. Visual Requirements Engineering for IT-Projects, 2007. `http://www.arcway.com/fileadmin/arcwaydateien/pdf/ARCWAY_VRE_Whitepaper.pdf`, visited 2007-09-15.

[Asc07] P. Aschenbrenner. Business Process driven Requirements Engineering, 2007. `http://www.arcway.com/fileadmin/arcwaydateien/pdf/BPRE_Whitepaper.pdf`, visited 2007-09-15.

[Kin06] E. Kindler. On the semantics of EPCs: Resolving the vicious circle. *Data Knowl. Eng*, 56(1):23–40, 2006.

[KNS92] G. Keller, N. Nüttgens, and A.-W. Scheer. Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). Technical Report Heft 89, Universität des Saarlandes, 1992. Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi).

[Kru04] P. Kruchten. *The Rational Unified Process*. Addison-Wesley, third edition, 2004.

[KW03] F. Keller and S. Wendt. FMC: An Approach Towards Architecture-Centric System Development. In *ECBS 2003*, pages 173–182. IEEE Computer Society, 2003.

[MLZ06] J. Mendling, K.B. Lassen, and U. Zdun. Transformation Strategies between between Block-Oriented and Graph-Oriented Process Modeling Languages. In F. Lehner, H. Nösekabel, and P. Kleinschmidt, editors, *Multikonferenz Wirtschaftsinformatik 2006 (MKWI 2006)*, volume 2, pages 297–312. GITO-Verlag Berlin, 2006. XML4BPM Track.

[Mol05] M. Molter. Die prozessorientierte Applikationslandschaft. In A.-W. Scheer, W. Jost, and K. Wagner, editors, *Von Prozessmodellen zu lauffähigen Anwendungen*, pages 151–172. Springer, 2005.

[Obj07] Object Management Group. *UML 2.1.1 Superstructure Specification*, February 2007.

[ODtHvdA07] C. Ouyang, M. Dumas, A.H.M. ter Hofstede, and Wil M.P. van der Aalst. Pattern-based translation of BPMN process models to BPEL web services. *International Journal of Web Services Research (JWSR)*, 2007.

[STA05] A.-W. Scheer, O. Thomas, and O. Adam. Process Modeling using Event-Driven Process Chains. In M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede, editors, *Process-Aware Information Systems*, pages 119–146. Wiley & Sons, 2005.

[Tab05] P. Tabeling. *Softwaresysteme und ihre Modellierung. Grundlagen, Methoden und Techniken*. Springer, Berlin, 2005.

[WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D.F. Ferguson. *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005.

[ZM05] J. Ziemann and J. Mendling. EPC-Based Modelling of BPEL Processes: a Pragmatic Transformation Approach. In *Proceedings of the 7th International Conference Modern Information Technology in the Innovation Processes of the Industrial Enterprises (MITIP 2005)*, Genova, Italy, September 2005.

Erzeugung detaillierter und ausführbarer Geschäftsprozessmodelle durch Modell-zu-Modell-Transformationen

Thomas Allweyer

Fachbereich Informatik und Mikrosystemtechnik
Fachhochschule Kaiserslautern
Standort Zweibrücken, Amerikastr. 1
66482 Zweibrücken
thomas.allweyer@fh-kl.de

Abstract: Es wird ein Ansatz vorgestellt, mit dem sich aus grobgranularen fachlichen Geschäftsprozessmodellen detaillierte, ggf. auch technisch verfeinerte, ausführbare Modelle in verschiedenen Zielnotationen erzeugen lassen. Hierzu wird ein der modellgetriebenen Software-Entwicklung vergleichbarer Ansatz gewählt, bei dem Modellelemente geeignet ausgezeichnet und dann mit Hilfe von parametrisierbaren Templates und Transformationsregeln in detaillierte Zielmodelle überführt werden. Vorteile sind eine Reduktion des Modellierungsaufwandes, einheitlichere und qualitativ hochwertigere Modelle und eine bessere und direktere Umsetzung fachlicher Modelle in ausführbare Prozessbeschreibungen.

1 Einleitung

Geschäftsprozesse werden zumeist auf verschiedenen Detaillierungsebenen modelliert. Die Spanne reicht von unternehmensweiten Prozesslandkarten mit groben Prozessclustern bis hin zu detaillierten Prozessbeschreibungen mit genauer Angabe von beteiligten Mitarbeiterrollen, Input- und Outputdaten, aufzurufenden Systemtransaktionen, usw. Einen sehr hohen Detaillierungsgrad weisen insbesondere Modelle auf, die von Business Process Management-Systemen (BPMS) direkt ausgeführt werden können. Hier müssen alle technischen Details, wie z. B. Parameter mit ihren Datentypen, die exakte Formulierung von Bedingungen usw. definiert sein, damit eine eindeutige und vollständige Ausführungsvorschrift für die verwendete Process Engine vorliegt.

Die meisten Methoden und Werkzeuge zur Geschäftsprozessmodellierung sehen daher die Möglichkeit einer hierarchischen Modellierung vor. In der Regel kann einer grobgranularen Funktion ein Modell eines Detailprozesses hinterlegt werden. Auch den Funktionen eines solchen Detailprozesses lassen sich wiederum detailliertere Prozessmodelle hinterlegen, usw. Auf diese Weise können prinzipiell beliebig tiefe Prozesshierarchien aufgebaut werden. In der Praxis findet man meist drei bis sechs Hierarchiestufen von der gesamten Unternehmensprozesslandkarte bis zum feinsten Detailmodell.

Um eine gewisse Einheitlichkeit zu erreichen, werden zumeist Modellierungskonventionen aufgestellt, in denen festgelegt wird, welche Details auf welcher Ebene modelliert werden sollen [Al05a].

Wie die Konsistenz zwischen über- und untergeordnetem Modell sichergestellt wird, ist ganz unterschiedlich geregelt: Z. T. gibt es die Möglichkeit, jedem Modellelement beliebige Modelle mit beliebigen Inhalten zu hinterlegen. In diesem Fall ist der Modellierer komplett selbst für das Zusammenpassen von unter- und übergeordnetem Modell verantwortlich. Im Falle von stärker formalisierten Modellierungsnotationen (z. B. UML Aktivitätsdiagramm[1] oder BPMN[2]) erzwingt die Notation die Einhaltung von Konsistenzregeln beim Verfeinern, indem In- und Output einer Funktion auch im hinterlegten Modell als In- und Output des Detailprozesses vorkommen müssen.

Das Erstellen sehr detaillierter Modelle ist recht aufwändig, weshalb eine Aufwands-/ Nutzen-Abschätzung erforderlich ist, um zu entscheiden, ob sich der Aufwand für einen sehr hohen Detaillierungsgrad lohnt, oder ob eine vergleichsweise grobe Modellierung ausreicht. Für manche Zwecke, wie genaue Analysen (z. B. mit Hilfe von Simulation), die Spezifikation von Informationssystemen, oder die Ausführung durch ein BPMS, sind hingegen zwangsläufig hinreichend detaillierte Modelle mit einer genauen Einhaltung von bestimmten Formalismen erforderlich, ohne die eine korrekte Simulation oder Ausführung nicht möglich sind. Problematisch ist, dass sich eine einheitliche Modellierung über sehr viele, zumeist von verschiedenen Modellierern erstellte Detailmodelle hinweg kaum sicherstellen lässt. Vergleichbare Sachverhalte werden daher oftmals sehr verschieden dargestellt. Auch die Sicherstellung der Qualität ist bei einer sehr großen Zahl detaillierter Modelle schwierig, wenn diese Modelle von Hand erstellt werden.

Erschwert wird die geschilderte Problematik, wenn Prozessmodelle für unterschiedliche Zwecke in verschiedenen Notationen und/oder Tools erstellt werden müssen. Häufig werden innerhalb desselben Unternehmens sowohl High End-Modellierungs-Suiten als auch einfache Grafiktools, spezielle Analyse- und Simulationswerkzeuge und die Modellierungskomponenten von Business Process Management-Systemen verwendet. Zumeist nutzen diese Tools verschiedene Notationen, oder sie verlangen auch bei gleichen Notationen die Einhaltung spezifischer Regeln oder die Verwendung von proprietären Notations-Erweiterungen. Hier ergibt sich die Problematik des Austausches von Modellen zwischen unterschiedlichen Modellierungstools sowie der Konvertierung zwischen verschiedenen Notationen (oder gar einer Übersetzung zwischen verschiedenen Modellierungs-Stilen innerhalb derselben Notation).

Hierfür existiert bereits eine Reihe von Ansätzen, z. B. zur Konvertierung von EPK in BPMN oder BPEL[3] und umgekehrt (z. B. [Ko05], [MZ05]). Diese Ansätze verfolgen die Zielsetzung einer möglichst guten eins-zu-eins-Übersetzung der Modellinhalte in die jeweilige Zielnotation. Andererseits ist eine solche in vielen Fällen gar nicht sinnvoll.

[1] UML steht für die in der objektorientierten Software-Entwicklung gebräuchliche „Unified Modeling Language“ [Om07], vgl. auch [Hi05], [Ru07].

[2] BPMN steht für die „Business Process Modeling Notation“, eine grafische Notation für die Modellierung von (insbesondere ausführbaren) Geschäftsprozessen [Om06a], vgl. auch [Ha05].

[3] BPEL steht für „Business Process Execution Language“, eine XML-basierten Sprache zur Spezifikation der Orchestrierung von Web Services im Rahmen ausführbarer Geschäftsprozesse [Oa07], vgl. auch [Ha05].

Wird ein Detailmodell eines bestimmten Prozesses nur für ein spezielles Tool, z. B. für die Ausführung in einem spezifischen BPMS, benötigt, so ist es nicht erforderlich, das betreffende Modell zunächst etwa als EPK zu erstellen und dann mit oftmals problembehafteten Konvertierungsmechanismen in die von dem BPMS benötigte Zielnotation zu überführen. Stattdessen genügt es in diesem Fall, das Detailmodell von Vornherein in der Modellierungskomponente des BPMS und der dort eingesetzten Notation zu erstellen. Wendet man dieses Verfahren (jedes Teil-Modell nur in einem Tool) konsequent an, so treten andererseits die oben diskutierten Probleme der Konsistenz und Einheitlichkeit durch die mangelnde Tool- und Methodenintegration in noch stärkerem Maße auf.

Werden verschiedene Methoden und Tools verwendet, so werden diese i. d. R. auch von verschiedenen Personen beherrscht und angewandt. Die Schwierigkeiten in der Kommunikation von Modellierern mit ganz unterschiedlichen Methoden- und Toolkenntnissen verschärfen die Problematik weiter. Leicht führt diese Situation dazu, dass die in den verschiedenen Tools erstellten Modelle völlig unabhängig voneinander (weiter) entwickelt werden, so dass überhaupt keine übergreifende Modellintegration und somit keine Gesamtsicht auf die Geschäftsprozesse mehr möglich ist.

Trotz der Entwicklung leistungsfähiger und flexibler Business Process Management-Systeme (BPMS) mit grafischen Modellierungsoberflächen bleibt in einem solchen Fall die viel beklagte Kluft zwischen Fachbereich und IT weitgehend bestehen: Die Entwicklung ausführbarer Prozesse erfolgt durch BPMS-Experten, der Bezug zu den fachlichen Geschäftsprozessmodellen fehlt weitgehend, oder muss durch aufwändige organisatorische Maßnahmen sichergestellt werden.

Die beiden genannten Problemfelder, nämlich die effiziente Erstellung qualitativ hochwertiger Detailmodelle einerseits, und die tool- und methodenübergreifende Modellintegration andererseits, sind eng miteinander verknüpft. Der im Folgenden zur Diskussion gestellte Ansatz ist dazu geeignet, einen Lösungsbeitrag für beide Problemfelder zu liefern.

Bei der Erarbeitung dieses Ansatzes wurde folgendermaßen vorgegangen: Zunächst wurden ausgehend von realen Modellen aus der Praxis Beispielmodelle für ausgewählte Sachverhalte erstellt. Anhand dieser wurden Art und Umfang der notwendigen Übergänge von grobgranularen Modellen zu detaillierten fachlichen und zu ausführbaren Modellen untersucht. Anschließend wurden existierende Ansätze zur Verfeinerung und Transformation von Modellen daraufhin analysiert, ob sie geeignet sind, die erforderlichen Übergänge zu schaffen. Als Ergebnis wurde ein Ansatz der Modell-zu-Modell-Transformation entwickelt, der sich an das Vorgehen der modellgetriebenen Software-Entwicklung anlehnt und die Grundidee auf die Transformation von Geschäftsprozessmodellen überträgt. Dieser Ansatz wurde mit Hilfe der entwickelten Beispielmodelle konkretisiert und validiert.

2 Modell-Transformationen in der Software-Entwicklung

Die Grundidee des vorgestellten Ansatzes ist es, durch Modell-zu-Modell-Transformationen detaillierte und ggf. ausführbare Prozessmodelle aus grobgranularen, abstrakteren Prozessmodellen zu generieren, anstatt diese manuell zu modellieren. Die Idee lehnt sich an aktuelle Ansätze zur modellgetriebenen Software-Entwicklung an, wie sie unter den Stichworten „Model Driven Architecture“ (MDA, [Om03]) oder „Model Driven Software Development“ (MDSD) diskutiert werden [St07], und überträgt diese in den Bereich der Geschäftsprozessmodellierung.

In der modellgetriebenen Software-Entwicklung werden Modelle verwendet, um daraus in einem oder mehreren Transformationsschritten Code zu generieren. Die Modelle werden typischerweise in UML oder einer domänenspezifischen Sprache (Domain Specific Language, DSL) erstellt. Sie weisen einen wesentlich höheren Abstraktionsgrad als der erzeugte Code auf. So wird etwa aus einer modellierten UML-Klasse mit ihren Attributen nicht nur eine entsprechende Java-Klasse generiert. Es können vielmehr zugleich auch ein Datenbankschema für die persistente Speicherung, der Zugriffscode für die Datenbank, der Code für die Darstellung und Manipulation entsprechender Objekte in der Benutzeroberfläche usw. generiert werden.

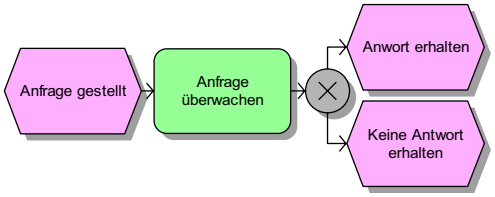
Hierzu werden Code-Templates verwendet. Ein Generator füllt diese Templates mit den Inhalten des Ausgangsmodells und fügt die generierten Code-Teile konsistent zusammen. Hierfür müssen geeignete Transformationsregeln definiert sein, die vom Generator ausgeführt werden. Aus ein und demselben Modell können durch Verwendung verschiedener Templates und Transformationsregeln ganz unterschiedliche Artefakte generiert werden, u. a. auch Code in diversen Programmiersprachen und für verschiedene technische Plattformen.

Z. T. ist es erforderlich, die Modellinhalte im Hinblick auf die Transformation speziell zu markieren, um z. B. festzulegen, für welche der modellierten Klassen ein persistenter Speicherungsmechanismus erzeugt werden soll. Die UML bietet für eine solche Auszeichnung beliebiger Modellelemente das Konstrukt des „Stereotyps“ an. Zusätzlich erforderliche Parameter für die Transformation können als so genannte „Tagged Values“ (Eigenschaftswerte) angegeben werden.

Bei den Transformationen muss es sich nicht ausschließlich um Transformationen von grafisch orientierten Modellen in Code handeln, vielmehr können beispielsweise auch Text-basierte Beschreibungen (z. B. XML-Dateien) transformiert oder erzeugt werden, und es sind auch Modell-zu-Modell-Transformationen möglich, wobei aus einem abstrakteren Modell ein konkreteres Modell erzeugt wird. Der von der Object Management Group (OMG) definierte MDA-Ansatz sieht explizit mehrere Modelltypen mit unterschiedlichen Abstraktionsgraden vor, wobei Computation Independent Models (CIM) nacheinander in Platform Independent Models (PIM) und schließlich in Platform Specific Models (PSM) transformiert werden können. Im Sinne der modellgetriebenen Software-Entwicklung kann der Übergang von grobgranularen zu detaillierteren Geschäftsprozessmodellen ebenfalls als Modell-zu-Modell-Transformation aufgefasst werden.

3 Beispiel: Überwachung von Anfragen

Die entwickelte Vorgehensweise wird anhand eines Beispiels erläutert. Häufig ist es erforderlich, die Reaktion auf eine Anfrage, eine Bestellung, eine versandte Rechnung o. ä. zu überwachen. Hierzu muss überprüft werden, ob die gewünschte Antwort, Lieferung oder Zahlung innerhalb einer gewissen Frist eingegangen ist. Gegebenenfalls muss nachgefragt oder gemahnt werden. Oftmals wird eine solche Überwachung pauschal als einzelne Funktion modelliert. Die wesentlichen Details können in Form eines beschreibenden Textes hinterlegt werden.



Beschreibung der Funktion „Anfrage überwachen“:

Falls keine Antwort eingeht, werden bis zu zwei Nachfragen gestellt: Nach 3 Wochen und ggf. nach 2 weiteren Wochen. Erfolgt auch dann noch keine Reaktion, so wird nach weiteren 2,5 Wochen eine Absage versandt.

Geht eine Antwort erst danach ein, so wird diese wegen verspäteten Eingangs abgelehnt.

Abbildung 1: „Anfrage überwachen“ in EPK sowie Beschreibung der Details

Abbildung 1 stellt eine solche pauschale Modellierung einer Überwachungsfunktion dar. In vielen Fällen mag diese Modellierung ausreichen, wenn sich die zuständigen Mitarbeiter bei der Prozessdurchführung eigenverantwortlich um die geeignete Realisierung des Detailablaufs kümmern.

Eine Möglichkeit, die Anfrage-Überwachung detailliert zu modellieren, ist in Abbildung 2 dargestellt. Bei der Darstellung dieses Ablaufs mit Hilfe der EPK ergeben sich eine Reihe methodischer Fragestellungen, insbesondere hinsichtlich der Darstellung von zeitlichen Ereignissen, der Auswertung von Bedingungen beim Eintreten von Ereignissen sowie dem Abbrechen von Teilprozessen. Diese Fragestellungen sind jedoch nicht Gegenstand des vorliegenden Beitrags und werden daher nicht weiter vertieft. Zielsetzung bei der Erstellung des gezeigten Modells war eine möglichst gute Verständlichkeit. Im Hinblick auf eine formale Analysierbarkeit, eine Simulation oder Ausführung des Prozesses wäre die gewählte Art der Darstellung eher ungeeignet.

Im Vergleich zur groben Modellierung in Abbildung 1 ist das detaillierte Modell wesentlich komplexer und daher zunächst auch unübersichtlicher und weniger leicht zu verstehen. Von daher wird man sich in fachlich orientierten Modellen, die vor allem dem Prozessverständnis dienen, häufig auf die grobe Darstellung beschränken.

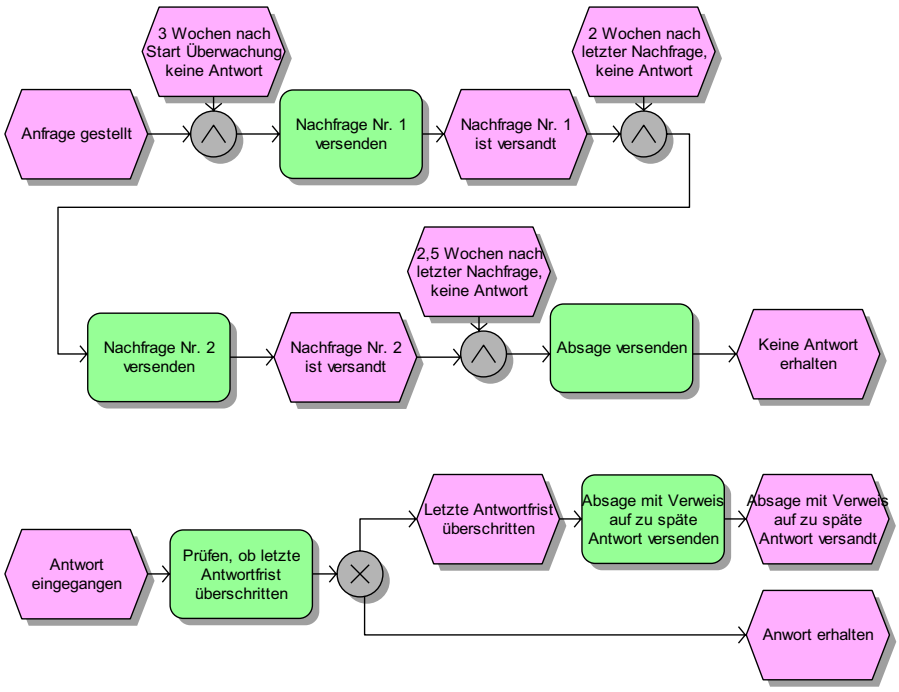


Abbildung 2: Ausführliche Modellierung der Überwachung einer Anfrage

Dennoch ist in vielen Fällen auch ein detailliertes Modell erforderlich, wenn z. B. eine detaillierte Handlungsanweisung für angelerntes Personal benötigt wird, wenn der genaue Prozessablauf aus Gründen gesetzlicher Nachweispflichten verbindlich dokumentiert sein muss, oder wenn der Prozess automatisiert von einer Process Engine ausgeführt werden soll. Im Falle der Automatisierung wird das ausführliche Modell i. d. R. nicht in Form einer EPK benötigt, sondern in einer Ausführungssprache, z. B. BPEL. Die im Folgenden vorgestellte Vorgehensweise lässt sich prinzipiell auch für andere Zielsprachen und –notationen als die EPK durchführen.

Das grobe Prozessmodell aus Abbildung 1 lässt sich auf verschiedene Arten in ein detailliertes Modell umsetzen. In Abbildung 2 wurden die in der Beschreibung enthaltenen Parameter (Zahl der Nachfragen, Fristen) direkt in Form des Kontrollflusses umgesetzt. Eine alternative Modellierung für die obere Hälfte der Abbildung 2 zeigt Abbildung 3. Hier wurden die Parameter aus der eigentlichen Kontrollflusslogik ausgelagert. Die in den Funktionen und Ereignissen genannte Maximalzahl und die Fristen müssen bei der Prozessdurchführung dem Informationsobjekt „Anfrage-Parameter“ entnommen werden.

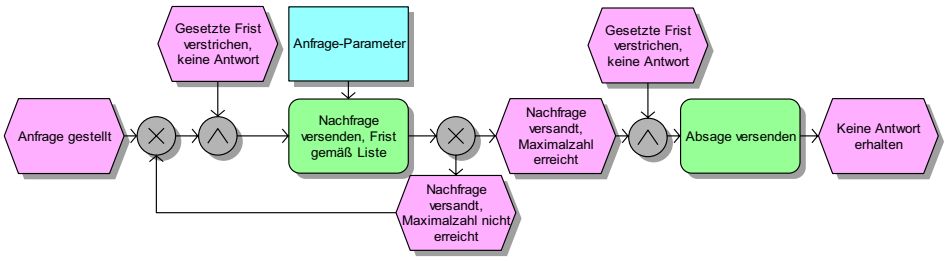


Abbildung 3: Alternatives Modell zur oberen Hälfte von Abbildung 2

Vorteil ist eine größere Flexibilität: Ändern sich lediglich die Parameter, so muss das Modell im Gegensatz zu Abbildung 2 nicht geändert werden. Vorteil der Variante aus Abbildung 2 hingegen ist die größere Aussagekraft des Modells an sich. Auch lassen sich reine Kontrollflusskonstrukte direkter in ausführbare Modelle umsetzen, wohingegen die Parameter-Abfrage und –Auswertung häufig zusätzlichen Realisierungsaufwand bedeuten.[4]

Im Folgenden soll gezeigt werden, wie ein derartiges Detail-Modell mit Hilfe einer geeigneten Modell-zu-Modell-Transformation automatisch generiert werden kann.

Ausgangspunkt der Transformation ist das Grobmodell aus Abbildung 1. Mit Hilfe einer Transformation soll daraus das detaillierte Modell aus Abbildung 2 erzeugt werden. Um eine entsprechend Transformation definieren zu können, muss im Ausgangsmodell ein bestimmtes Muster verwendet werden. Für das hier diskutierte Beispiel der Überwachung einer Anfrage o. ä. ist dieses Muster relativ simpel (linke Seite von Abbildung 4): Es handelt sich um eine Funktion mit zwei alternativen Folge-Ereignissen, wobei das eine Folge-Ereignis das positive Ergebnis einer erfolgreichen Überwachung repräsentiert, das andere das negative Ergebnis, d. h. es ist keine Antwort o. ä. eingegangen. Das auslösende Ereignis spielt hierbei keine wesentliche Rolle, es repräsentiert lediglich den vorangehenden Teil des Prozessmodells, das zum Starten der Funktion führt. Hierbei könnte es sich etwa auch um mehrere durch Konnektoren miteinander verbundene Ereignisse handeln.

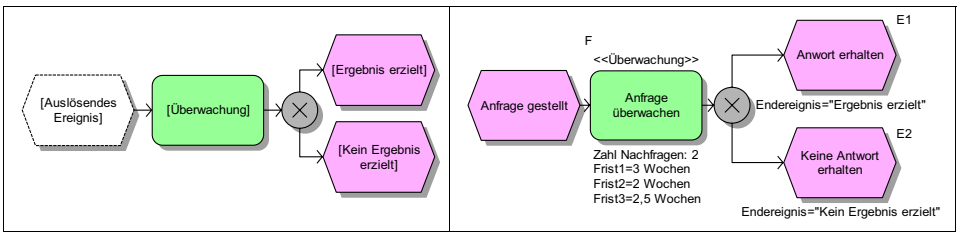


Abbildung 4: Muster „Überwachung“ (links), Anwendung des Musters in einer EPK (rechts)

[4] Die Auslagerung von Regeln aus der eigentlichen Prozesslogik lässt sich mit Hilfe von Business Rules Management-Systemen realisieren, vgl. z. B. [De05], [SG06], [Wa07].

Auf der rechten Seite von Abbildung 4 ist die konkrete Anwendung dieses Musters in einer als Ausgangsmodell dienenden EPK dargestellt. Das herkömmliche Modell aus Abbildung 1 muss um einige transformationsspezifische Informationen ergänzt werden. Zum einen muss angegeben werden, welches Muster verwendet werden soll. Hierfür wurde das aus der UML stammende Konstrukt der „Stereotypen“ verwendet, die dazu dienen, Modellelemente nach beliebigen Kriterien zu kategorisieren. In Beispiel wurde die Funktion F mit dem Stereotyp «Überwachung» ausgezeichnet. Weiterhin muss eine Zuordnung der verschiedenen Modellelemente zu den betreffenden Elementen des Musters erfolgen. Im Beispiel wurden hierzu die beiden Ereignisse E1 und E2 mit einem ebenfalls der UML entlehnten Tagged Value „Endereignis“ versehen. Auch die benötigten Parameter (Zahl der Nachfragen und Fristen) sind in Form von Tagged Values angegeben.

Abbildung 5 zeigt, was zur Transformation des derart angereicherten Ausgangsmodells noch benötigt wird. Zunächst ist es sinnvoll, das Ausgangsmodell zu validieren. Hierdurch wird sichergestellt, dass es richtig strukturiert ist und alle erforderlichen Informationen enthält, um erfolgreich transformiert zu werden. In Abbildung 6 sind entsprechende Validierungsregeln für die Verwendung des Stereotyps «Überwachung» dargestellt.

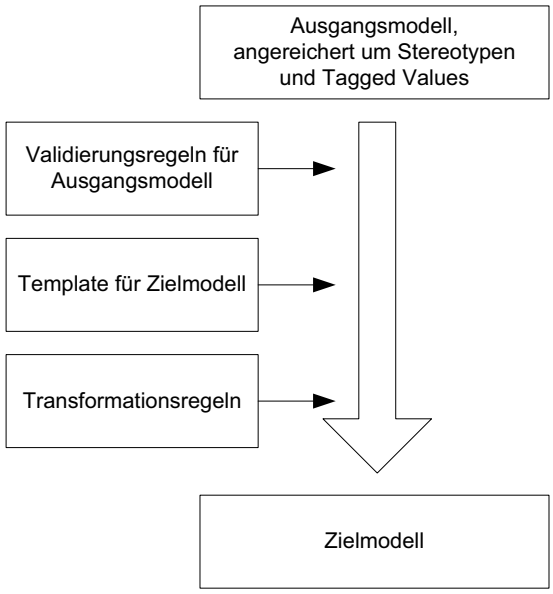


Abbildung 5: Modell-zu-Modell-Transformation

```
- Die Funktion mit dem Stereotyp „Überwachung“ (hier als F
  bezeichnet) verfügt über:
  1. einen Tagged Value „Zahl Nachfragen“ mit einem
     Integer-Wert > 1
  2. So viele Tagged Values „Frist [i]“ wie die Zahl der
     Nachfragen (Typ Zeitdauer).
- Auf die Funktion folgt eine XOR-Verknüpfung, die genau zwei
  Folge-Ereignisse hat.
- Eines der beiden Folge-Ereignisse (hier als E1 bezeichnet)
  hat einen Tagged Value „Endereignis“ mit dem Wert „Ergebnis
  erzielt“.
- Das andere der beiden Folge-Ereignisse (hier als E2
  bezeichnet) hat einen Tagged Value „Endereignis“ mit dem
  Wert „Kein Ergebnis erzielt“.
```

Abbildung 6: Validierungsregeln für die Verwendung des Stereotyps «Überwachung»

Basis für die Erstellung des Zielmodells ist das in Abbildung 7 gezeigte Template. Es enthält bereits die Struktur und die wesentlichen Elemente des Zielmodells. Eine Reihe von Platzhaltern muss jedoch noch mit konkreten Werten gefüllt werden. Außerdem enthält es eine Sequenz, die ggf. mehrfach wiederholt werden kann. Hier wurde die ausführliche Modellierung der Überwachung gewählt (vgl. Abbildung 2). Alternativ hätte auch die kompaktere Darstellung nach Abbildung 3 verwendet werden können.

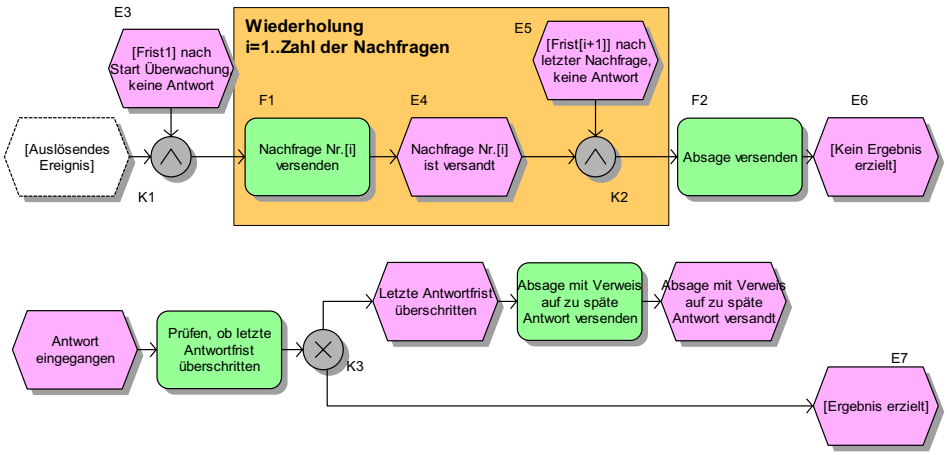


Abbildung 7: Template für die detaillierte Modellierung einer Überwachung

Bei der eigentlichen Transformation wird dieses Template mit Hilfe der in Abbildung 8 dargestellten Transformationsregeln und des Ausgangsmodells (rechte Seite von Abbildung 4) angepasst und in das in Abbildung 2 gezeigte Zielmodell überführt.

- Verbinde die in F eingehende(n) Kante(n) mit Konnektor K1 statt mit F.
- Entferne F.
- Ersetze in E3 „[Frist1]“ durch den Wert des Tagged Value Frist1 von F.
- Setze i=1. Solange i kleiner oder gleich der Zahl der Nachfragen ist, wiederhole Folgendes:
  - Erzeuge neue Objekte und Kanten für die Vorlagen im Rechteck „Wiederholung“.
  - Wenn i=1: verbinde K1 mit F1, sonst verbinde K2 der vorangehenden Wiederholung mit F1.
  - Ersetze in F1 und E4 jeweils „[i]“ durch den Wert von i.
  - Ersetze in E5 „Frist[i+1]“ durch den Wert des Tagged Value Frist[i+1] von F.
- Verbinde K2 der letzten Wiederholung mit F2.
- Verbinde F2 mit E2 statt mit E6.
- Verbinde K3 mit E1 statt mit E7.

Abbildung 8: Transformationsregeln für den Stereotyp «Überwachung»

Die dargestellten Transformationsregeln ersetzen die ursprüngliche Funktion im Ausgangsmodell durch die neu erzeugten Modellelemente. Es wird also kein neues Modell erzeugt, sondern das vorhandene Modell modifiziert. Im vorliegenden Fall könnte man die Transformationsregeln auch derart verändern, dass stattdessen ein neues Modell erzeugt würde, das der Funktion „Anfrage überwachen“ hinterlegt wird. Es sind jedoch andere Fälle denkbar, bei denen das Ausgangsmuster beispielsweise mehrere Funktionen umfasst, so dass das erzeugte Modell nicht als Detaillierung einer einzelnen Funktion dargestellt werden kann.

4 Transformationen beim Übergang zu ausführbaren Modellen

Soll ein auf fachlicher Ebene modellierter Geschäftsprozess durch ein Business Process Management-System (BPMS) unterstützt werden, so ist die Überführung in eine ausführbare Prozessbeschreibung erforderlich. Liegt das Prozessmodell nicht in einer von der Process Engine des verwendeten Systems direkt ausführbaren Notation vor, so ist zunächst eine Transformation in eine andere Notation oder Beschreibungssprache notwendig, z. B. von EPK zu BPMN oder BPEL.

Wie in der Einleitung bereits ausgeführt, geht es nicht darum, eine möglichst identische Konvertierung der Inhalte aus einer Notation in eine andere durchzuführen. Stattdessen soll aus einem grobgranularen, fachlichen Modell (hier in Form einer EPK) ein detailliertes, ausführbares Modell in der Notation des Zielsystems erzeugt werden.

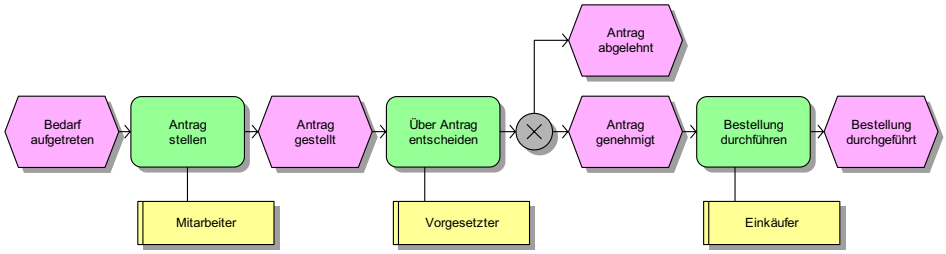


Abbildung 9: Fachliche EPK eines Ablaufs, der durch Workflow unterstützt werden soll

Abbildung 9 zeigt das als EPK modellierte fachliche Modell eines einfachen Beschaffungsprozesses. Soll dieser Prozess nun mit Hilfe des Systems Intalio|BPMS workflowunterstützt ablaufen, so ist eine Überführung in das BPMN-Modell nach Abbildung 10 erforderlich.

Bereits auf den ersten Blick wird deutlich, dass eine reine 1:1-Umsetzung der EPK-Logik in die BPMN nicht ausreicht. Zur Modellierung von Workflows, die mit menschlichen Benutzern interagieren, verlangt Intalio die Aufteilung in einen System-Pool für die Modellierung des eigentlichen Kontrollflusses sowie einen separaten Pool für jede Mitarbeiter-Rolle. Eine einzelne von einem Mitarbeiter durchzuführende Funktion muss in bis zu vier BPMN-Tasks aufgeteilt werden.

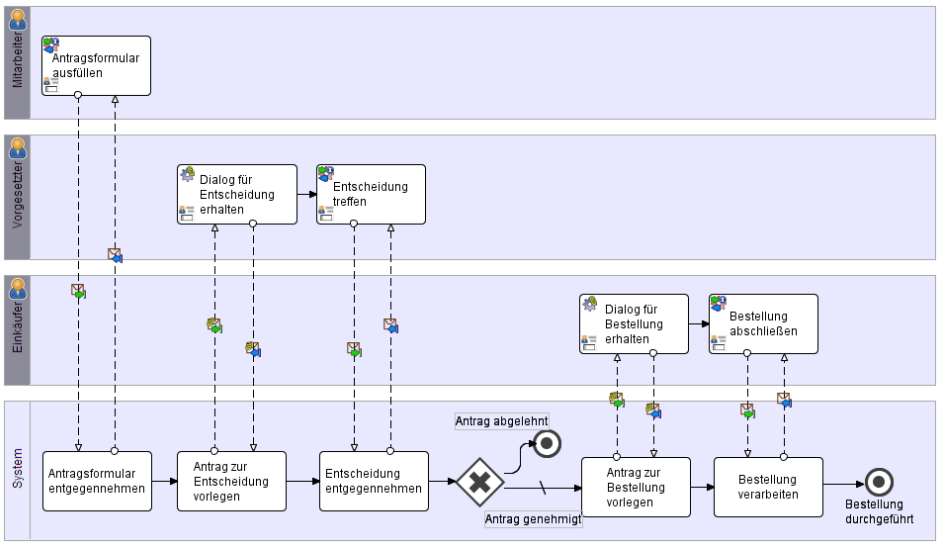


Abbildung 10: Umsetzung der EPK aus Abbildung 9 in ein ausführbares BPMN-Modell (für Intalio|BPMS)

Beispielsweise werden für die Funktion „Über Antrag entscheiden“ vier Tasks benötigt. Der erste Task „Antrag zur Entscheidung vorlegen“ übermittelt die benötigten Daten, der zweite Task erstellt den Dialog und trägt die Aufgabe in die Task-Liste der betreffenden

Rolle ein, der dritte Task „Entscheidung treffen“ beinhaltet das Ausfüllen und Absenden des Dialogs, und der vierte Task „Entscheidung entgegennehmen“ ist schließlich für die Verarbeitung des Ergebnisses zuständig.

Mit Hilfe des vorgestellten Ansatzes lässt sich eine derartige Aufteilung einer Funktion in vier durch Kontroll- und Nachrichtenflüsse verbundene Tasks automatisieren. Hierzu kann ein geeigneter Stereotyp „Workflow-Funktion“ definiert werden, für den ein BPMN-Template mit den vier Funktionen sowie geeignete Transformationsregeln existieren müssen.

Darauf aufbauend kann ein Stereotyp für die Entscheidungsfunktion mit genau zwei möglichen Ergebnissen definiert werden, der die Umsetzung in die BPMN-Logik regelt. Z. B. muss sichergestellt werden, dass die Entscheidungsfunktion eine Boole’sche Variable als Ergebnis liefert, die mit Hilfe geeignet formulierter Bedingungen steuert, welche der aus dem Gateway ausgehenden Kontrollflusskanten gewählt wird.

Im Gegensatz zu einer 1:1-Umsetzung von EPK-Modellen in eine ausführbare Notation können mit dem vorgestellten Ansatz in einem Schritt auch notwendige Verfeinerungsschritte beim Übergang von einem fachlichen in ein detailliertes, ausführbares Modell durchgeführt werden.

Dieser Schritt ist vergleichbar mit der im Software Engineering erfolgten Entwicklung weg von der Code-Generierung in traditionellen CASE-Tools hin zur Model Driven Architecture (MDA). Traditionelle CASE-Tools boten eine 1:1-Umsetzung grafischer Modelle (z. B. UML-Modelle) in Programmcode. Dies bedeutete, dass die Modelle auf dem gleichen Abstraktions- und Detaillierungsgrad wie der endgültige Programmcode sein mussten. Der MDA-Ansatz hingegen erlaubt die Generierung umfangreicheren Programmcodes aus abstrakteren Modellen. Hierdurch werden insbesondere bei Mehrfachverwendung der Modelle Produktivitäts- und Qualitätsvorteile erzielt. Mit Hilfe verschiedener Templates und Transformationsregeln kann z. B. aus ein und demselben Modell Code für ganz verschiedene Zielplattformen erzeugt werden.

Dies kann mit dem vorgestellten Ansatz ebenso für die Prozessausführung erreicht werden. Beispielsweise können fachliche Prozesse teilweise von unterschiedlichen Process Engines ausgeführt werden. Neben eigenständigen BPMS können innerhalb eines Prozesses z. B. auch Workflow-Komponenten eines Content Management-Systems oder eines Anwendungssystems zum Einsatz kommen.

Herkömmlich führt der doch erhebliche manuelle Aufwand bei der Umsetzung fachlicher Modelle in ausführbare Modelle häufig dazu, dass die ausführbaren Modelle von Grund auf neu entwickelt, in der Ausführung getestet und verbessert werden, und fachliches und ausführbares Modell völlig voneinander entkoppelt werden.

5 Anwendungsmöglichkeiten fachlicher Stereotypen

In der modellgetriebenen Software-Entwicklung werden Stereotypen eher für implementierungsbezogene Kategorisierungen verwendet, z. B. um anzugeben, dass Objekte einer bestimmten Klasse persistent gespeichert werden sollen. Bei der Software-Generierung wird für diese Klasse dann der notwendige Code für die betreffenden Datenbankzugriffe erzeugt.

Das in Abschnitt 4 vorgestellte Beispiel des Übergangs von fachlichen zu ausführbaren Modellen entspricht dieser Idee. Anstelle von Code in einer herkömmlichen Programmiersprache wird eine Implementierung in Form eines von einer Process Engine ausführbaren Modells erzeugt.

Das in Abschnitt 3 vorgestellte Beispiel des Stereotyps «Überwachung» bezieht sich hingegen auf fachliche Aspekte. Die prinzipielle Vorgehensweise ist dennoch die gleiche. In beiden Fällen wird nämlich eine Konkretisierung und Detaillierung des betreffenden grobgranularen, abstrakteren Modellelements vorgenommen, einmal in Form von Programmcode (bzw. ausführbaren Prozessmodellen), im anderen Fall in Form von detaillierteren fachlichen Anweisungen. Das Detailmodell kann im übertragenen Sinne als eine „organisatorische Implementierung“ der übergeordneten Funktion aufgefasst werden.

Die durch Stereotypen in der Software-Generierung bezeichneten implementierungsbezogenen Konzepte, sind recht klar definier- und abgrenzbar, und sie werden vielfach verwendet. Beispiele sind neben der bereits genannten Persistenz Stereotypen für verschiedene Elemente der Benutzungsoberfläche, für diverse Aufgaben der Verwaltung von Objekten durch eine Middleware (z. B. Kommunikation, Sicherheit, Verteilung) oder für Schnittstellen.

Die für die Transformation fachlicher Prozessmodelle in ausführbare Modelle verwendbaren Stereotypen ergeben sich aus den Funktionalitäten der Process Engines. Neben von Benutzern ausführbaren Workflow-Tasks sind beispielsweise Konzepte wie der Aufruf komplett automatisierter Funktionen, der Versand und Empfang von Nachrichten, Vertreterregelungen, Eskalationsmechanismen, langlaufende Transaktionen oder Ausnahmebehandlungen abzubilden.

Für die Generierung detaillierter fachlicher Modelle sind vergleichbar gut abgrenzbare Muster weniger offensichtlich. Beispiele könnten sein:

- Die Behandlung von Stornierungen und Abbrüchen, die an beliebigen Stellen eines Teilprozesses auftreten können.
- Die Verarbeitung von Änderungen, z. B. eines erteilten Auftrages, an beliebigen Stellen eines Teilprozesses.
- Überprüfungen und Kontrollschritte mit anschließenden Freigaben und ggf. Überarbeitungen.

- Prozesse, deren Bearbeitungsobjekte unter gewissen Umständen aufgeteilt werden können, wobei anschließend jede Teilmenge für sich bearbeitet wird, z. B. das Splitten von Aufträgen in verschiedene Teillieferungen.

Die Identifikation und Analyse derartiger Konzepte wird auch Gegenstand weiterer Forschung sein. Insbesondere ist zu prüfen, in wieweit sich für die einzelnen Konzepte Muster finden lassen, die tatsächlich in verschiedenen Prozesskontexten sinnvoll wiederverwendet werden können.

Es ist auch möglich, ein und dieselbe grobgranulare fachliche Funktion einerseits in Form eines ausführbaren Modells zu implementieren und andererseits organisatorisch mit Hilfe eines detaillierten fachlichen Modells umzusetzen. Hierzu kann das Grobmodell unverändert bleiben, es müssen lediglich die Templates und Transformationsregeln ausgetauscht werden. Insofern lassen sich die beiden vorgestellten Konzepte auch gemeinsam anwenden, wenn etwa nur ein Teil der Prozesse mit Hilfe eines BPMS automatisiert wird, ein anderer Teil hingegen manuell ausgeführt wird.

6 Zusammenfassung und Ausblick

Im vorliegenden Beitrag wurde ein Ansatz vorgestellt, detaillierte und ggf. ausführbare Prozessmodelle durch Modell-zu-Modell-Transformationen aus grobgranularen Prozessmodellen zu generieren. Hierzu werden Templates und Transformationsregeln verwendet. Die Anwendung dieses Ansatzes wurde anhand der Generierung einer Detail-EPK für die Überwachung einer Anfrage sowie anhand der Umsetzung einer einfachen EPK in ein ausführbares BPMN-Modell demonstriert.

Der Ansatz ist dazu geeignet, den Aufwand für die Erstellung von detaillierten Modellen zu reduzieren. Die Einheitlichkeit, Konsistenz und Qualität der Detailmodelle wird erhöht, und der Übergang von fachlichen zu eher technischen, ausführbaren Modellen wird erleichtert.

Wie bereits erläutert lehnt sich der Ansatz an Verfahren zur modellgetriebenen Software-Entwicklung an. Hierzu wurden u. a. einige Elemente der MDA übernommen. Das vorgestellte Verfahren setzt jedoch das MDA-Konzept der OMG nicht komplett um. So wurde beispielsweise auf die Spezifikation von Ziel- und Quellsprache (EPK bzw. BPMN mit den beschriebenen Erweiterungen) mit Hilfe eines MOF-konformen Metamodells[5] und die Repräsentation und Verarbeitung der Modelle mit Hilfe von XMI[6] verzichtet. Ebenso wurde zunächst keine konkrete Transformationssprache verwendet, sondern eine Formulierung der Transformationsregeln in Form von Pseudocode vorgenommen. Beides wäre jedoch ohne weiteres möglich und könnte etwa im Vorfeld einer Software-Implementierung der beschriebenen Transformationen hilfreich sein.

[5] MOF steht für „Meta Object Facility“, einen OMG-Standard zur Beschreibung von Metamodellen [Om06b].
[6] XMI steht für „XML Metadata Interchange“, einen OMG-Standard für den Austausch von Modellen [Om05].

Im Sinne der Referenzmodelladaption lässt sich die dargestellte Vorgehensweise auch als Instanziierung von Referenzmodellen interpretieren[7]: Hierbei kann das in Abbildung 7 gezeigte Template als Referenzmodellbaustein aufgefasst werden, der mit Hilfe des vorgestellten Transformationsmechanismus an den konkreten Anwendungskontext angepasst wird. Die Besonderheit des vorgestellten Ansatzes liegt einerseits in eben diesem, der modellgetriebenen Software-Entwicklung angelehnten Mechanismus begründet, andererseits in der Verwendung von grobgranularen Prozessmodellen mit Stereotypen und Tagged Values zur Spezifikation der durchzuführenden Instanziierung.

Das vorgestellte Verfahren ist zunächst unabhängig von den verwendeten Modellierungssprachen und kann damit insbesondere auch unabhängig von bestimmten Interpretationen der durch eine EPK ausgedrückten Semantik durchgeführt werden. Bei der Erstellung konkreter Templates und Transformationsregeln wird es in der Praxis erforderlich sein, gewisse Annahmen über die zugrunde liegende Semantik zu machen. Ersteller von Templates und Transformationsregeln einerseits sowie Modellierer grobgranularer Prozesse (und damit Anwender der Transformationen) andererseits müssen über ein einheitliches Verständnis der jeweils intendierten Bedeutung verfügen. Dies muss nicht unbedingt über eine formale Definition der Semantik erfolgen, in der Praxis kann eine informale Erläuterung, z. B. anhand von Beispielen, für den Modellierer sogar nützlicher sein.

Ein zentraler Gegenstand der weiteren Arbeiten wird insbesondere die Validierung des Ansatzes und seiner praktischen Anwendbarkeit sein. Hierzu werden weitere, komplexere Anwendungsfälle identifiziert und in Form von Templates und Transformationsregeln umgesetzt. Hierbei soll der Ansatz auch erweitert werden, um nicht nur eine reine Umsetzung des reinen Kontrollflusses zu ermöglichen, sondern auch weitere Modellinhalte wie organisatorische Informationen, Input- und Outputdaten u. ä. zu transformieren. Im Zusammenspiel mit modellgetriebener Software-Entwicklung im eigentlichen Sinne könnte dann beispielsweise nicht nur der Kontrollfluss für ein Workflow Management-System generiert werden, sondern auch ein gewisser Anteil an Anwendungslogik, wie z. B. Benutzerdialoge für die Bearbeitung der einzelnen Arbeitsschritte (vgl. [Al05b]). Hierbei sollen geeignete Muster sowie Modellierungsrichtlinien für die Transformations-geeignete fachliche Modellierung entwickelt werden. Nicht zuletzt werden entsprechende Transformatoren softwaretechnisch implementiert, wobei vorhandene Modellierungswerkzeuge und Transformationsframeworks einbezogen werden sollen.

Literaturverzeichnis

[Al05a] Allweyer, T.: Geschäftsprozessmanagement – Strategie, Entwurf, Implementierung, Controlling. W3L-Verlag, Herdecke Bochum, 2005.

[Al05b] Allweyer, T.: Maßgeschneiderter Methodeneinsatz für die Entwicklung betriebswirtschaftlicher Software. In: Scheer, A.-W.; Jost, W.; Wagner, K. (Hrsg.): Von Prozessmodellen zu lauffähigen Anwendungen. Springer, Berlin Heidelberg New York, 2005, S. 173-195.

[7] Vgl. die Übersicht über Verfahren zur Referenzmodelladaption in [Be04], S. 42f.

[Be04] Becker, J.: Data Warehousing und Referenzmodellierung: Ever Tried to Build a House Without a Construction Plan? In: Becker, J. et al. (Hrsg.) Working Paper No. 1. European Research Center for Information Systems (ERCIS). Münster 2004, S. 37-48.

[De05] Debevoise, T.: Business Process Management with a Business Rules Approach. Implementing the Service Oriented Architecture. Business Knowledge Architects, Roanoke, 2005.

[Ha05] Havey, M.: Essential Business Process Modeling. O'Reilly, Sebastopol, 2005.

[Hi05] Hitz, M. et al.: UML @ Work. 3. Aufl., dpunkt Verlag, Heidelberg, 2005.

[Ko05] Kopp, O.: Abbildung von EPKs nach BPEL anhand des Prozessmodellierungswerkzeugs Nautilus. Diplomarbeit. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2005.

[MZ05] Mendling, J.; Ziemann, J.: Transformation of BPEL Processes to EPCs. In: Nüttgens, M.; Rump, F. J. (Hrsg.): EPK 2005 Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. 4. Workshop der Gesellschaft für Informatik e.V. (GI) und Treffen ihres Arbeitskreises "Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)", Hamburg, 2005, S. 41-53.

[Oa07] OASIS (Hrsg.): Web Services Business Process Execution Language Version 2.0. OASIS, Billerica, 2007.

[Om03] OMG (Hrsg.): MDA Guide Version 1.0.1. OMG, Needham, 2003.

[Om05] OMG (Hrsg.): MOF 2.0/XMI Mapping Specification, v.2.1. OMG, Needham, 2005.

[Om06a] OMG (Hrsg.): Business Process Modeling Notation Specification. Version 1.0. OMG, Needham, 2006.

[Om06b] OMG (Hrsg.): Meta Object Facility (MOF) Core Specification. Version 2.0. OMG, Needham, 2006.

[Om07] OMG (Hrsg.): Unified Modeling Language: Superstructure. Version 2.1.1. OMG, Needham, 2007.

[Ru07] Rupp, C.; Queins, S.; Zengler, B.: UML 2 glasklar. Praxiswissen für die UML-Modellierung. 3. Aufl., Hanser, München Wien 2007.

[SG06] Schacher, M.; Grässle, P.: Agile Unternehmen durch Business Rules. Der Business Rules Ansatz. Springer, Berlin Heidelberg New York, 2006.

[St07] Stahl, T.; Völter, M.; Efftinge, S.: Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management. 2. Auflage. Dpunkt Verlag, Heidelberg, 2007.

[Wa07] Wagner, J.: Agile Geschäftsprozesse durch Integration von Geschäftsregeln. IDS Scheer AG, Saarbrücken, 2007.

On the Degree of Behavioral Similarity between Business Process Models

Jan Mendling[†], Boudewijn van Dongen[‡], Wil van der Aalst[‡]
[†]Queensland University of Technology
126 Margaret Street, QLD 4000 Brisbane, Australia
j.mendling@qut.edu.au
[‡]Eindhoven University of Technology
PO Box 513, NL-5600 MB Eindhoven, The Netherlands
b.f.v.dongen@tue.nl, w.m.p.v.d.aalst@tue.nl

Abstract: Quality aspects become increasingly important while business process modeling is used in a large-scale enterprise setting. In order to facilitate a storage without redundancy and an efficient retrieval of relevant process models in model databases it is required to develop a theoretical understanding of how a degree of behavioral similarity can be defined. In this paper we address this challenge in a novel way. We use *causal footprints* as an abstract representation of the behavior captured by a process model, since they allow us to compare models defined in both formal modeling languages like Petri nets and informal ones like EPCs. Based on the causal footprint derived from two models we calculate their similarity based on the established vector space model from information retrieval. We illustrate this concept with an example from the SAP Reference Model and present a prototypical implementation as a plug-in to the ProM framework.

1 Introduction

Business process modeling is gaining increasing attention as a basis for the development of large-scale enterprise information systems. In this context, business process models can either be used as a formalization of requirements that guide the implementation, as input for code generation in a model-driven architecture, or as executable templates on a dedicated process engine defined e.g. with BPEL. All these three scenarios have in common that various quality issues have to be considered in order to facilitate *storage without redundancy* and an *efficient retrieval of models.* These two goals have in common that there require a theoretical understanding of how the degree of behavioral similarity can be formalized. Based on such a concept it would be easy to identify those models that should be integrated with others for maintainability reasons and to rank models that belong to the answer set of a query. Such a functionality would be in particular helpful to analysts who have to integrate the operations of two enterprises that engage in a merger. We will see in the related work section that the current state of the art does not provide a general solution to this problem.

The analysis of behavioral similarity of business process models is complicated by two

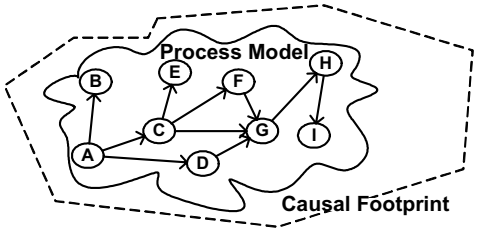


Figure 1: A causal footprint provides a characterization of the process model such that the behavior of the process is enclosed [DMA06].

problems. First, there is a plethora of languages for business process modeling. Even though there is some overlap between the different languages as shown by pattern-based evaluations [AHKB03], there are striking differences and many subtle semantical issues that complicate matters. For example, a concept such as the "synchronizing merge" [AHKB03], also know as the OR-join in languages such as BPMN and EPCs, can be interpreted in different ways and often leads to semantical paradoxes as shown in [ADK02, Kin06]. Second, and even worse, most business process modeling languages have no formal semantics or their semantics is only well-defined for a restricted subset. Still, redundancy and retrieval issues matter also for those business process models. In this paper, we address these problems in a novel way.

The classical approach to comparing process models is to construct a state space or enumerate all possible traces and then compare the models based on this. Trace equivalence and bisimulation are typical notions used to compare formal models on such basis. Unfortunately, these equivalence notions typically only work for models that have formal semantics and have finite behavior (e.g., the number of traces or states needs to be bounded). Moreover, such notions provide a "yes/no" answer rather than the degree of similarity. Therefore, we abstract from the precise behavior using a concept that is called *causal footprint* [DMA06]. The idea of a causal footprint is that it describes a set of conditions on the order of activities that hold for the process model. These conditions can be used to reason about *similarity* of footprints. Figure 1 illustrates that the causal footprint gives an approximation of the process behavior in terms of conditions that every process instance has to obey to. In [DMA06] it has been shown that causal footprints can be used to reason about the soundness of process models. Although causal footprints abstract from the detailed behavior, it is still possible to find certain errors (e.g., particular types of deadlocks).

In this paper, we use *causal footprints for measuring similarity*. Unlike our earlier work [DMA06] we do not look at verification but at the degree of similarity between two models. When comparing two models we compare the corresponding causal footprints using the so-called vector model used in information retrieval [SWY75, BYRN99].

Throughout this paper, we use Event-driven Process Chains (EPCs) and Petri nets to show the applicability of causal footprints for both formal and conceptual business process modeling languages. This choice is motivated by the fact that EPCs are widely used for the documentation of business processes, e.g. in the SAP reference model. Furthermore, Petri nets are a well-understood formalism for the modeling of business processes and

have been used for the formalization of a variety of languages and standards (e.g. BPMN, BPEL, XPDL, UML activity diagrams). Against this background the paper is structured as follows. First, Section 2 gives an introduction to EPCs and Petri nets and shows how both can be mapped to Causal Footprints. Then, Section 3 presents a novel concept for calculating the degree of behavioral similarity between two process models based on their causal footprints and inspired by concepts from information retrieval. Section 6 discusses our approach in the light of related work before Section 7 concludes the paper with an outlook on future research.

2 Preliminaries

In the introduction, we mentioned two process modeling languages, namely Petri nets and EPCs. In this section, we introduce these modeling languages informally and provide formal definitions for both languages (only syntax). Furthermore, we introduce the concept of a causal footprint and show how to derive causal footprint from EPCs and Petri nets by just considering their structure. Most modeling languages are graph based. Therefore, we start by introducing some notation specifically for directed graphs.

Definition 2.1. (Pre-set and Post-set)

Let $G = (N, E)$ be a directed graph consisting of a set of nodes N and a set of edges $E \subseteq N \times N$. Moreover, let $n \in N$ be a particular node. We define ${}^{G}\bullet n = \{m \in N \mid (m, n) \in E\}$ as the pre-set and $n{\bullet}^{G} = \{m \in N \mid (n, m) \in E\}$ as the post-set of n with respect to the graph G. If the context is clear, the superscript G may be omitted, resulting in $\bullet n$ and $n\bullet$.

2.1 Petri nets and EPCs

Petri nets can be used to specify (possibly concurrent) processes in an unambiguous manner. Since the language has formal and executable semantics, processes modelled in terms of a Petri nets can be executed by an information system. In fact most workflow engines have adopted the basic constructs present in the Petri net formalism. For an elaborate introduction to Petri nets, the reader is referred to [DE95,Mur89,RR98]. A Petri net consists of two types of node elements (cf. Figure 2): *Transitions* typically correspond to either an activity which needs to be executed, or to a "silent" step that takes care of routing. A transition is drawn as a rectangle. *Places* are used to define the preconditions and postconditions of transitions. A place is drawn as a circle. Transitions and places are connected through directed arcs in such a way that (i) places and transitions have at least one incident edge and (ii) in every path, transitions and places alternate (no place is connected to a place and no transition is connected to a transition). For completeness sake, we mention that the Petri nets we use in this paper correspond to a classic subclass of Petri nets, namely workflow nets [Aal98], which are tailored towards workflow modeling and analysis.

Definition 2.2. (Workflow net)

$\omega = (P, T, F)$ is a workflow net (or WF-net [Aal98]) if:

- P is a finite set of places,
- T is a finite, non empty set of transitions, such that $P \cap T = \emptyset$ and $T \neq \emptyset$,
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation of the net,
- there exists exactly one $p_i \in P$, such that $| \bullet p_i| = 0$,
- there exists exactly one $p_f \in P$, such that $|p_f \bullet | = 0$,
- all places and transitions are covered by the paths from p_i to p_f.

In addition to Petri nets, we also use EPCs to illustrate our ideas. EPCs provide an intuitive modeling language to model business processes. EPCs were introduced by Keller, Nüttgens, and Scheer in 1992 [KNS92]. It is important to realize that the language is not intended to be a *formal* specification of a business process. Instead, it serves mainly as a means of communication. EPCs are extensively used in large-scale enterprise modeling projects. One prominent example of a publicly available model is the SAP reference model [CKL97, KT98]. An EPC consists of three types of node elements (cf. Figure 2): *Functions* correspond to an activity (task, process step) which needs to be executed. A function is drawn as a box with rounded corners. *Events* describe the situation before and/or after a function is executed. An event typically corresponds to the pre- or post-condition of a function. Events are drawn as hexagons. *Connectors* can be used to connect functions and events to specify the flow of control. There are three types of connectors: $\wedge$ (AND), $\times$ (XOR) and $\vee$ (OR). Connectors are drawn as circles, showing the type in the center. Functions, events, and connectors can be connected with edges in such a way that (i) events have at most one incoming edge and at most one outgoing edge, but at least one incident edge (i.e. an incoming or an outgoing edge), (ii) functions have precisely one incoming edge and precisely one outgoing edge, (iii) connectors have either one incoming edge and multiple outgoing edges, or multiple incoming edges and one outgoing edge, and (iv) in every path, functions and events alternate (ignoring intermediate connectors).

Definition 2.3. (Event-driven Process Chain)

$\varepsilon = (F, E, C_{and}, C_{xor}, C_{or}, A)$ is an EPC if:

- F is a finite set of functions,
- E is a finite set of events,
- $C = C_{and} \cup C_{xor} \cup C_{or}$ is a finite set of connectors, such that $|C| = |C_{and}| + |C_{xor}| + |C_{or}|$,
- $A \subseteq ((F \cup E \cup C) \times (F \cup E \cup C))$ is the flow relation of the net, such that:
  - for all $f \in F$ there is one $(f, x) \in A$ and one $(x, f) \in A$,
  - for all $e \in E$ there is at most one $(e, x) \in A$ and at most one $(x, e) \in A$,
  - for all $c \in C$ there is either one $(c, x) \in A$ and more than one $(x, c) \in A$, or one $(x, c) \in A$ and more than one $(c, x) \in A$,
  - on all paths, functions and events alternate.

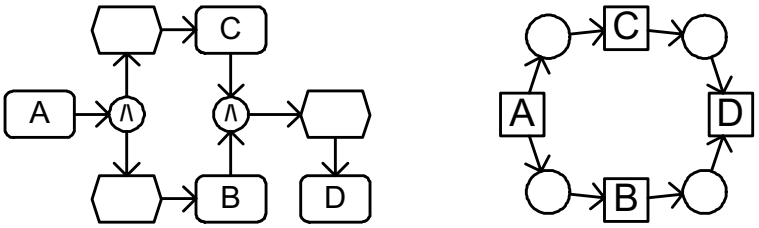


Figure 2: An EPC (left) and a Petri net (right) describing a process thats starts with A, followed by the parallel execution of B and C, and ends with D.

Note that in this section, we only provide an abstract syntax of Petri nets and EPCs and do not give any semantics. The reason is that we are not interested in the precise semantics. As illustrated by Figure 1 we aim for a causal footprint which is independent of specific and/or refined semantical interpretations (e.g., the OR-join in EPCs).

2.2 Causality graphs

As stated in the introduction, we look at a similarity measure for business processes, based on the high-level concept of causal footprints. At the basis of causal footprints lie *causality graphs*.[1] A causality graph captures the intented behavior that is captured in the process model. It is important to realize that these causality graphs do not capture the entire process model as such, in fact they are merely a footprint of the control flow in the given process model, i.e. they describe the approximate behavior of a system at a very high level.

Definition 2.4. (Process behavior/case)

Let T be a set of activities, and let Φ_T be a process containing these activities. The behavior of the process Φ_T is defined as the set $W \subseteq T^*$, where T^* is the set of all sequences that are composed of zero of more tasks from T. A $\sigma \in W$ is called a *case*, i.e. a possible execution of the process. To denote an activity at a specific index in σ, we use $\sigma[i]$, where i is the index ranging from 1 to $|\sigma|$.

We have now formalized the behavior of a process and each of the modeling languages of Section 2 is intended to capture this behavior in a structured way. As we stated before, we will look at processes at a high level. Therefore, we introduce the *causality graph*, representing a footprint of the process.

Definition 2.5. (Causality Graph)

Let N be a set of activities. We define a causality graph $G = (N, F_{lb}, F_{la})$, where:

- N is a finite set of *nodes* (activities),

[1] Note that this paper adopts the concept of a causality graph from [DMA06]. However, unlike in [DMA06] this concept is used for measuring similarity rather than verification.

- $F_{lb} \subseteq (\mathcal{P}(N) \times N)$ is a set of *look-back links*[2],
- $F_{la} \subseteq (N \times \mathcal{P}(N))$ is a set of *look-ahead links*.

When a causality graph is used to describe a process, it should be interpreted in the following way. For each *look-ahead link*, we say that the execution of the source of that link leads to the execution of at least one of the targets of that link, i.e., if $(a, B) \in F_{la}$, then any execution of a is followed by the execution of some $b \in B$. A look-ahead link is denoted as a bullet with one or more outgoing arrows. Furthermore, for each *look-back link*, the execution of the target is preceded by at least one of the sources of that link, i.e., if $(A, b) \in F_{lb}$, then any execution of b is preceded by the execution of some $a \in A$. The notation of a look-back link is a bullet with one or more incoming arrows. Note that we do not give any information about when in the future or past executions took place, but only that they are there. This way of describing a process is similar to the work presented in [EJL$^+$99]. However, by splitting up the semantics in the two different directions (i.e. forward and backward), causal footprints are more expressive. With footprints you can for example express the fact that task `A` is always succeeded by `B`, but that `B` can also occur before `A`, which is typically hard to express in other languages.

If a causality graph indeed describes a process like this, we call it a *causal footprint*. We formalize this concept using the notion of cases.

Definition 2.6. (Causal Footprint)

Let T be a set of activities, Φ_T be a process with behaviour W. Furthermore, let t_i and t_f be such that $t_i, t_f \notin T$ and $t_i \neq t_f$. Furthermore, let $G = (T \cup \{t_i, t_f\}, F_{lb}, F_{la})$ be a causality graph. (For notational purposes, we say that for all $\sigma \in W$ with $n = |\sigma|$, it holds that $\sigma[0] = t_i$ and $\sigma[n+1] = t_f$, i.e. we add artificial starts and ends to each trace). We say that G is a *causal footprint* graph of Φ_T, denoted by $G \in \mathcal{F}_{\Phi_T}$, if and only if:

1. For all $(a, B) \in F_{la}$ holds that for each $\sigma \in W$ with $n = |\sigma|$, such that there is a $0 \leq i \leq n+1$ with $\sigma[i] = a$, there is a $j : i < j \leq n+1$, such that $\sigma[j] \in B$,
2. For all $(A, b) \in F_{lb}$ holds that for each $\sigma \in W$ with $n = |\sigma|$, such that there is a $0 \leq i \leq n+1$ with $\sigma[i] = b$, there is a $j : 0 \leq j < i$, such that $\sigma[j] \in A$,

It is clear from Definition 2.6 that a causal footprint is not unique, i.e., different processes can have common footprints. For example, $G = (T \cup \{t_i, t_f\}, \emptyset, \emptyset)$ is the causal footprint of any process having activities T. Therefore, we aim at footprints that are more informative without trying to capture detailed semantics. By deriving a transitive closure, a causal footprint can be extended to be a more informative causal footprint.

In [DMA06], the transitive closure of a causal footprint was defined for the soundness analysis of these footprints. In this paper, we use the same *causal closure* to describe similarities between process models.

Definition 2.7. (Causal Closure)

Let $G = (N, F_{lb}, F_{la})$ be a causality graph. We define $G^* = (N, F^*_{lb}, F^*_{la})$ to be the causal closure of G, where F^*_{lb} and F^*_{la} are the smallest possible sets, such that:

[2]With $\mathcal{P}(N)$, we denote the powerset of N, i.e. $N' \in \mathcal{P}(N)$ if and only if $N' \subseteq N$ and $N' \neq \emptyset$.

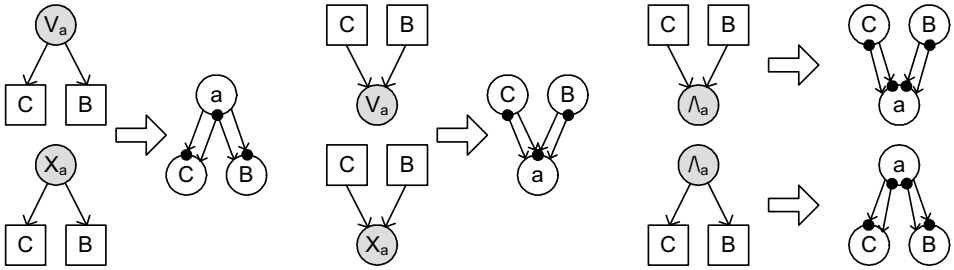


Figure 3: Mapping of EPCs to causal footprints.

1. $(a, B) \in F_{la}$ implies that $(a, B) \in F^*_{la}$,
2. $(A, b) \in F_{lb}$ implies that $(A, b) \in F^*_{lb}$,
3. $(a, B) \in F^*_{la}$ implies that for all $N' \subseteq N$ holds that $(a, B \cup N') \in F^*_{la}$,
4. $(A, b) \in F^*_{lb}$ implies that for all $N' \subseteq N$ holds that $(A \cup N', b) \in F^*_{lb}$,
5. $(a, B) \in F^*_{la}$, $b \in B$ and $(b, C) \in F^*_{la}$, implies that $(a, (B \setminus \{b\}) \cup C) \in F^*_{la}$,
6. $(B, c) \in F^*_{lb}$, $b \in B$ and $(A, b) \in F^*_{lb}$, implies that $((B \setminus \{b\}) \cup A, c) \in F^*_{lb}$,

The rules for causally closing a causal graph obviously apply to a causal footprint as well. More importantly, the causal closure of a causal footprint is a causal footprint again [DMA06] and we refer to it as a *footprint closure*. The fact that a causal footprint of a process is not unique is irrelevant for the work presented in this paper. Any property that can be derived to hold on a causal footprint of a process, holds on the process itself. The better the causal footprint of a process is, i.e. the more information it contains, the more properties we are able to deduce. As an example, again consider the graph that only has the activities of a process as nodes and no edges. This graph is a causal footprint of any process, but it does not contain any information about the process.

2.3 Deriving Causal Footprints

In this section, we present algorithms to derive causal footprints of EPCs and Petri nets. This is done in two stages. First, the models are translated to causality graphs in such a way that these graphs contain all elements of the modeling languages and only causalities that can be derived locally (i.e., immediate predecessor and successor relationships). These graphs are then transitively closed and projected onto the subset of interesting elements, thus leading to causal footprints.

Definition 2.8. (EPC to causal graph)

Let $\varepsilon = (F, E, C_{and}, C_{xor}, C_{or}, A)$ be an EPC, with the set of modeling elements $N' = F \cup E \cup C_{and} \cup C_{xor} \cup C_{or}$, the set of initial elements $N_i = \{n \in N' \mid \overset{\varepsilon}{\bullet} n = \emptyset\}$ and the set of final elements $N_f = \{n \in N' \mid n \overset{\varepsilon}{\bullet} = \emptyset\}$. We define a causality graph $G_\varepsilon = (N, F_{lb}, F_{la})$ as follows:

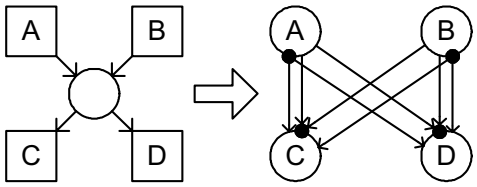


Figure 4: Mapping for Petri net.

- $N = N' \cup \{n_i, n_f\}$, where $n_i, n_f \notin N'$ and $n_i \neq n_f$,
- $F_{la} =$
  $\{(a, \{b\}) \in N \times \mathcal{P}(N) \mid a \in F \cup E \cup C_{and} \wedge b \in a\overset{\varepsilon}{\bullet}\} \cup$
  $\{(a, B) \in N \times \mathcal{P}(N) \mid a \in C_{or} \cup C_{xor} \wedge B = a\overset{\varepsilon}{\bullet}\} \cup$
  $\{(n_i, N_i)\} \cup \{(a, \{n_f\}) \mid a \in N_f\}$,
- $F_{lb} =$
  $\{(\{a\}, b) \in \mathcal{P}(N) \times N \mid b \in F \cup E \cup C_{and} \wedge a \in \overset{\varepsilon}{\bullet} b\} \cup$
  $\{(A, b) \in \mathcal{P}(N) \times N \mid b \in C_{or} \cup C_{xor} \wedge A = \overset{\varepsilon}{\bullet} b\} \cup$
  $\{(N_f, n_f)\} \cup \{(\{n_i\}, a) \mid a \in N_i\}$.

Definition 2.8 gives an algorithm to derive a causality graph from an EPC. This is illustrated by Figure 3. It is important to note that the causality graph is derived based on the structure of the model, i.e., there is no need to first construct the set of cases. In some cases it is useful to restrict the causality graph to a subset of nodes. For EPCs for example, we might only be interested in its functions and connectors. By taking the causal closure of the causality graph and projecting it onto the functions and connectors (i.e. removing all events and all edges related to these events), we obtain a causal footprint of the process modelled by the EPC. For this, we first define a projection of a causal graph.

Definition 2.9. (Causal Graph Projection)

Let $G = (N, F_{lb}, F_{la})$ be a causal graph and let N' be a set of nodes, such that $N' \subseteq N$. We define the projection $G' = (N', F'_{lb}, F'_{la})$ of G onto N', such that ${F_{lb}}' = F_{lb} \cap (\mathcal{P}(N') \times N')$ and ${F_{la}}' = F_{la} \cap (N' \times \mathcal{P}(N'))$.

As we stated before, the translation rules from an EPC to a causal graph are given under the assumption that the EPC is sound. However, soundness would imply that there are clear executable semantics, which is not always the case. Therefore, our translation rules do not require explicit semantics. The only requirement for our rules to hold is that soundness should be defined in such a way that a synchronizing connector is considered to fire eventually.

Similar to EPCs, we provide a translation for WF-nets to causality graphs and show that the result is a causal footprint for the modelled process (cf. Figure 4).

Definition 2.10. (Workflow net to causal footprint)

Let $\omega = (P, T, F)$ be a WF-net, with the set of modeling elements $N' = P \cup T$, the initial place $p_i \in P$ and final place $p_f \in P$. We derive a causality graph $G_\omega = (N, F_{lb}, F_{la})$ as

follows:

- $N = T \cup \{n_i, n_f\}$, where $n_i, n_f \notin N'$and $n_i \neq n_f$,
- $F_{la} =$
  $\{(n, N') \in N \times \mathcal{P}(N) \mid \exists_{p \in P \setminus \{p_f\}}\ n \in {}^{\omega}\bullet p \wedge N' = p^{\omega}\bullet\} \cup$
  $\{(n_i, p_i{}^{\omega}\bullet)\} \cup$
  $\{(t, \{n_f\}) \mid t \in {}^{\omega}\bullet p_f\}$,
- $F_{lb} =$
  $\{(N', n) \in \mathcal{P}(N) \times N \mid \exists_{p \in P \setminus \{p_i\}}\ n \in p^{\omega}\bullet \wedge N' = {}^{\omega}\bullet p\} \cup$
  $\{({}^{\omega}\bullet p_f, n_f)\} \cup$
  $\{(\{n_i\}, t) \mid t \in p_i{}^{\omega}\bullet\}$,

As shown in [DMA06] causal footprints can be used to detect problems such as deadlocks. However, the causal footprint also provides a compact but useful characterization of the corresponding process. Therefore, the concept can be used to compare processes which is useful when searching for similar models or for quantifying the gap between some desired model and a given model.

3 Similarity of Business Process Models

In many scenarios it is important to compare business process models with respect to the behavior they specify, e.g. in case of a merger of two companies that have models for similar business processes, or when looking for a web services closest to the own internal processes. In this section, we consider the causal footprints of two models as an input to calculate similarity. This has several advantages compared to related proposals. First, causal footprints include information about the order of activities beyond direct succession. The full closure also looks at indirect dependencies and, therefore, it is more powerful than the approach in [AAW06], since there only direct connections are used for structural similarity. Furthermore, causal footprints are robust with respect to problems such as termination or finiteness of state space which affect true/false behavioral similarity measures, such as trace-equivalence and (branching) bisimilarity. We calculate similarity directly based on the causal footprint of two models instead of considering their explicit behavior. In order to do so we use techniques originating from information retrieval [SWY75, BYRN99].

3.1 Similarity based on the Vector Model

In information retrieval the degree of similarity between a document and a query plays a very important role for ranking the returned documents according to their relevance. For calculating similarity, we use the well-known *vector model* [SWY75, BYRN99] which is one of the basic techniques used for information filtering, information retrieval, and the indexing of web pages. Its classical application is to determine the similarity between a query and a document. The original vector space model proposed by Salton, Wong, and

Yang in [SWY75] attaches weights based on term frequency to the so-called "document vector". We use a more liberal interpretation, where other weights are possible. However, to explain the basic mechanism we use terms originating from the domain of information retrieval, i.e., terms like "document collection", a set of "terms", and a set of "weights" relating to the terms. Later we will provide a mapping of these terms to causal footprints.

The *document collection* contains a set of documents. Each of these documents is considered to be a list of *terms* which are basically the words of the document. The union of all terms of all documents is then used to describe each document as a vector. For one specific document an entry in the vector represents that the term associated with the vector position of this entry is included in the document. In a simple case the occurrence of a term can be indicated by a one and the non-occurrence with a zero, however there is also the option to assign *weights* to terms in order to address the fact that they differ in relevance. A common choice is to use one divided by the number of occurrences of a term throughout all documents of the document collection as a weight which has the effect that scarcely used terms get a higher weight. A query can also be considered as a document, i.e., a list of terms.

The similarity between a query and a document is then calculated based on their vector representation as the cosine of the angle between the two vectors [SWY75, BYRN99]. Calculating this degree of similarity for each document provides a mechanism to rank them according to their relevance for the query.

Our proposal for determining the similarity of two business process models builds on the vector model and causal footprints. We consider causal footprints of two processes $G_1 = (N_1, F_{1,lb}, F_{1,la})$ and $G_2 = (N_2, F_{2,lb}, F_{2,la})$ as input for the calculation. In order to apply the vector model, we have to define (1) the document collection, (2) the set of terms, and (3) the set of weights.

The document collection includes two entries corresponding to the two causal footprints that need to be compared. We will refer to these as the first and the second causal footprint (i.e., G_1 and G_2).

The set of terms is build from the union over nodes, look back, and look ahead links of the two causality closures. We define $\Theta = N_1 \cup F_{1,lb} \cup F_{1,la} \cup N_2 \cup F_{2,lb} \cup F_{2,la}$ as the set of terms and $\lambda : \Theta \to \{1, 2, \ldots |\Theta|\}$ as an indexing function that assigns a running number to each term, i.e., the set of all elements appearing in the two footprints are enumerated. (Note that we implicitly assume all sets of nodes and links to be disjoint in a single model.)

The relevance of each term is closely related to the number of tasks from which it is built. Consider for example two look ahead links $x_{la} = (a, \{g\}) \in F_{la}$ and $y_{la} = (a, \{b, c, d, e, f\}) \in F_{la}$. x_{la} refers to only two tasks: a and g. y_{la} refers to six tasks (a through f). It seems obvious that the look ahead links with fewer tasks are more informative and therefore more important. To address this we use weights depending on the number of tasks involved in a look-ahead/back link.

The weights are determined using the size of the relations. If $\theta \in \Theta$ is a single node (i.e. $\theta \in N_1 \cup N_2$), then we define the weight of θ as $w_\theta = 1$. Furthermore, since the number of potential look ahead and look back links depends upon the powerset of

nodes, is seems natural to use exponentially decreasing weights. Therefore, for all links $\theta \in \Theta$, we define the weight of a link $w_\theta = 1/(2^{|\theta|-1})$, where $|\theta|$ denotes the number of tasks in the link, i.e. for the two look ahead links $x_{la} = (a, \{g\})$ and $y_{la} = (a, \{b, c, d, e, f\})$, we get $w_{x_{la}} = 1/(2^{2-1}) = 0.5$ and $w_{y_{la}} = 1/(2^{6-1}) = 0.03125$.

Using the document collection, the set of terms and the weights presented above, we define the document vectors, which we call *footprint vectors*.

Definition 3.1. (Footprint vectors)

Let $G_1 = (N_1, F_{1,lb}, F_{1,la})$ and $G_2 = (N_2, F_{2,lb}, F_{2,la})$ be two causal footprints, with Θ the set of terms and $\lambda : \Theta \rightarrow \mathbb{N}$ an indexing function. We define two footprint vectors, $\overrightarrow{g_1} = (g_{1,1}, g_{1,2}, \ldots g_{1,|\Theta|})$ and $\overrightarrow{g_2} = (g_{2,1}, g_{2,2}, \ldots g_{2,|\Theta|})$ for the two models as follows. For each element $\theta \in \Theta$, we say that for each $i \in \{1, 2\}$ holds that

$$g_{i,\lambda(\theta)} = \begin{cases} 0 & \text{if } \theta \notin (N_i \cup F_{i,lb} \cup F_{i,la}) \\ w_\theta = \dfrac{1}{2^{|\theta|-1}} & \text{if } \theta \in (F_{i,lb} \cup F_{i,la}) \\ w_\theta = 1 & \text{if } \theta \in N_i \end{cases}$$

Using the two footprint vectors, we can define the similarity between two footprints as the cosine of the angle between these two vectors.

Definition 3.2. (Footprint similarity)

Let $G_1 = (N_1, F_{1,lb}, F_{1,la})$ and $G_2 = (N_2, F_{2,lb}, F_{2,la})$ be two causal footprints, with Θ the set of terms and $\lambda : \Theta \rightarrow \mathbb{N}$ an indexing function. Furthermore, let $\overrightarrow{g_1}$ and $\overrightarrow{g_2}$ be the corresponding footprint vectors. We say that the similarity between G_1 and G_2, denoted by $sim(G_1, G_2)$ is the cosine of the angle between those vectors, i.e.

$$sim(G_1, G_2) = \frac{\overrightarrow{g_1} \times \overrightarrow{g_2}}{|\overrightarrow{g_1}| \cdot |\overrightarrow{g_2}|} = \frac{\sum_{j=1}^{|\Theta|} g_{1,j} \cdot g_{2,j}}{\sqrt{\sum_{j=1}^{|\Theta|} g_{1,j}^2} \cdot \sqrt{\sum_{j=1}^{|\Theta|} g_{2,j}^2}}$$

The value of $sim(G_1, G_2)$ ranges from 0 (no similarity) to 1 (equivalence). In this paper, we do not elaborate on this formula. If one accepts the weights that we associate to the "terms" in a causal footprint, then the cosine of the angle between these two vectors provides a generally accepted way to quantify similarity [SWY75, BYRN99].

The similarity $sim(G_1, G_2)$ between footprints can be calculated for any two footprints G_1 and G_2. However, the best results will be obtained if the causal footprint is a footprint closure, since then the most information is obtained. To illustrate this, we again take two examples from the SAP reference model.

4 Application to the SAP reference model

To show the practical applicability in real life, we applied our approach in the context of the SAP reference model. In this section, we illustrate the calculation of business process model similarity based on the vector model and causal footprints using this reference model.

Since reference models should be comprehensive sets of models serving as a guide when modelling large information systems, we feel that they should satisfy two properties:

- The reference model should contain models that are correct, i.e. there should not be any errors in these models.
- The reference model should be searchable, i.e. one should not only be able to search for models of a specific process (such as "purchasing" or "invoice management"), but one should also be able to search through the database to find a model that matches a given model as close as possible. This is in particular needed in merger situations where similar operations of two enterprise have to be integrated.

The first requirement has been extensively discussed in [DJVVA, ADMV06, MADV06b, MADV06a, MVD+07] and it was mentioned in [DMA06], where we showed how causal footprints can be used to find errors in EPCs.

A first step towards satisfying the latter of the two requirements is provided in this paper, i.e. we present a way to compare the behaviour of models based on their structure. In the work presented here, it is assumed that it is easy to map the activities of one model to the activities of another model (e.g. by assuming that they are the same objects, or have the same label). In our application, this mapping was done manually since the labels of functions not always coincided completely (e.g. "Order creation" vs. "refurbishment order creation", etc.).

In [DJVVA], we presented a guided model selection to find errors in the SAP reference model. In that paper, we were guided by one specific function called "Order Execution" that appeared in several EPCs (seven in total). In this paper, we take the same guide, i.e. we look at the seven EPCs containing the function "Order Execution" and we pairwise compare their similarity. The seven EPCs that are investigated are presented in Table 1.

For each of the seven models, we calculated the causal footprint and we compared them using our similarity metric of Definition 3.2. It is important to note that the seven models do not all meet the first requirement stated above, i.e. most of them even show syntactical problems by having events linked to events. However, since events are not reflected in a causal footprint, this does not influence our results.

Table 2 shows the result of our comparison. Again note that at some points, we manually had to make mappings from the functions in one model to functions in the other model. Figure 5 shows how this mapping was made in the implementation in ProM (cf. Section 5). It shows that a mapping was made from "order creation" to "maintenance order creation", from "order execution" to "order execution" and from "order release" to "maintenance order release". Furthermore, for model 2 on the left hand side, the functions "material planning", "order permit" and "order printing" were not mapped to any of the functions

Table 1: The seven EPCs containing the function "order execution" in the SAP reference model.

1. plant maintenance / breakdown maintenance processing / order / order,
2. plant maintenance / planned maintenance processing / order / order,
3. plant maintenance / project based maintenance processing / order / order,
4. plant maintenance / refurbishment processing in plant maintenance / order / order,
5. quality management / test equipment management / maintenance order / maintenance order,
6. quality management / test equipment management / service order / service order,
7. treasury / forex spot forward and swap transactions / transaction processing / transaction processing.

in the other model. Similarly, the functions "overall completion confirmation", "order settlement" and "order completion" in model 5 on the right hand side were not mapped to any of the functions in model 2.

The results presented in Table 2 give us valuable insights into the SAP reference model. First, they show that models containing the function "order execution" within the same module (i.e. "plant maintenance", "quality management" and "treaury") are exactly the same (in terms of behaviour). The question arises whether this is desirable, i.e. should there be so many copies of exactly the same model?

Furthermore, Table 2 also shows that the similarity between models in different modules is rather low. This implies that although these models contain a reference to exactly the same function "order execution", the contexts in which this function is executed are rather different and hence that the SAP reference model basically contains three different functions, which are all labelled "order execution".

Finally, it should be noted that the mere fact that two EPCs are 26.4% similar in itself is not too informative. However, in a querying scenario this measure can be used to rank several process models of the result set (for example, when looking for a model that is similar to model 1 of Table 1 the models 2,3 and 4 are more likely candidates than the

Table 2: Similarity between EPCs containing the function "order execution" in the SAP reference model.

	1	2	3	4	5	6	7
1	100 %	100 %	100 %	100 %	26.4 %	26.4 %	5.28 %
2		100 %	100 %	100 %	26.4 %	26.4 %	5.28%
3			100 %	100 %	26.4 %	26.4 %	5.28%
4				100 %	26.4 %	26.4 %	5.28%
5					100 %	100 %	5.01%
6						100 %	5.01%
7							100 %

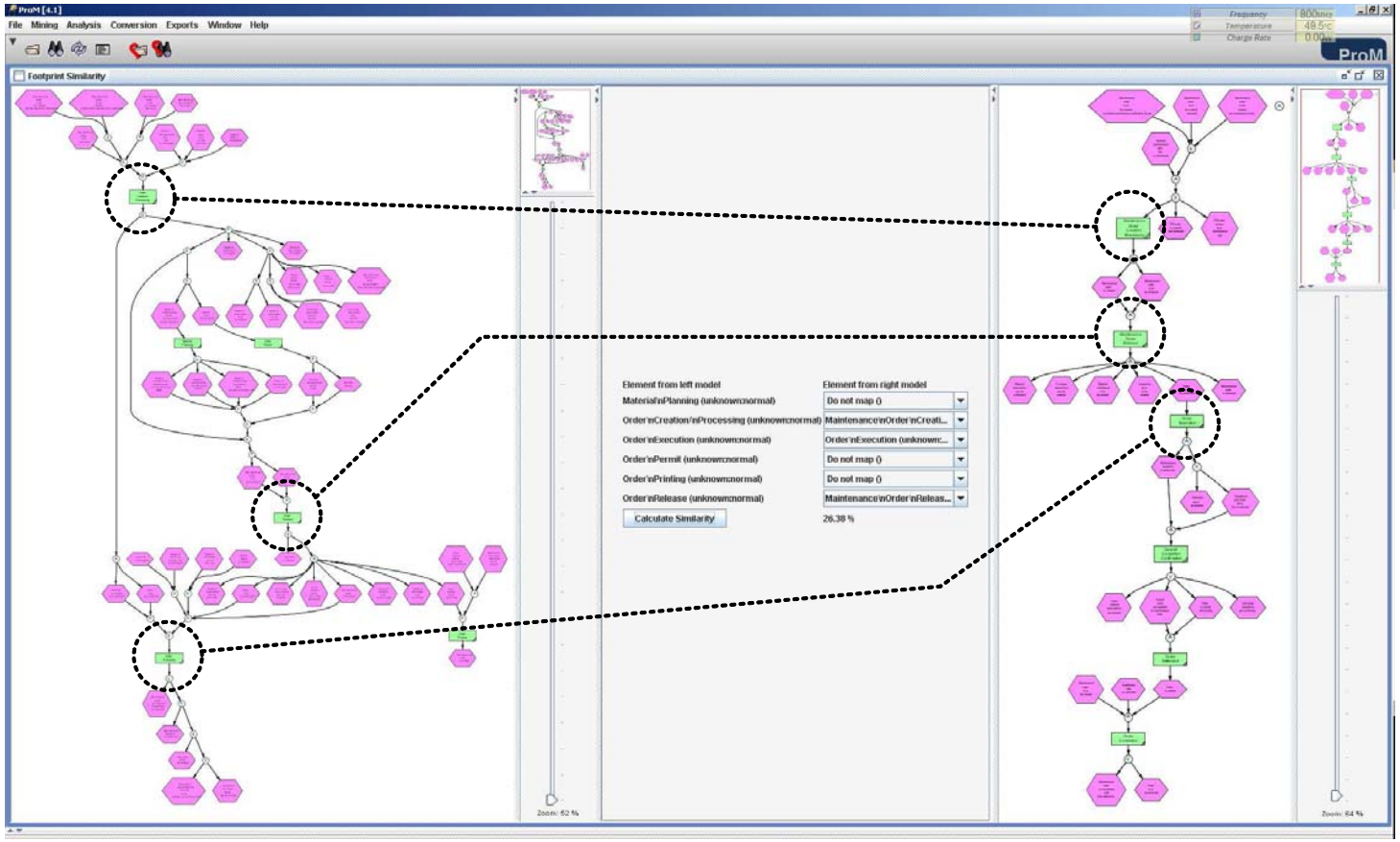


Figure 5: Comparison of two SAP models in ProM: $sim(G_2, G_5) = 0.2638$.

models 5,6 and 7) and in an merger scenario this value might be high enough to advocate an integration of the process models as is the case with models 1, 2, 3 and 4 and models 5 and 6.

5 Tool support

The (Pro)cess (M)ining framework *ProM* has been developed as a completely plug-able environment for process mining and related topics in the context of business process analysis. It can be extended by simply adding plug-ins, i.e., there is no need to know or to recompile the source code. Currently, more than 150 plug-ins have been added [DAV+05]. ProM is open-source and can be downloaded from `http://www.processmining.org`.

In the context of this paper, we developed an analysis plug-in for the comparison of causal footprints. This analysis plug-in, takes two causal footprints (or models that can be converted to causal footprints, e.g. Petri nets and EPCs, using the already available plug-ins presented in [DMA06]) and calculates their similarity. The plug-in requires the user to first map the elements of one model onto the elements of the other model. To support this step the plug-in suggests a mapping based on the labels of corresponding elements. Furthermore, by default, the analysis plug-in considers only functions for EPCs and only visible transitions for Petri nets, by projecting the footprints on these elements before calculating similarity. Figure 5 shows the comparison of two EPCs in ProM.

For the practical application of our results it is important to note that the ProM framework is currently capable of reading and writing EPCs to AML (native to the Aris Toolset), EPML (a standard for storing EPCs), VDX (the XML format of Visio), and the Aris graph format used by Aris PPM. For Petri nets, multiple formats are available, including the standard PNML format. ProM also supports other languages such as YAWL and BPEL. Since causal footprints are language independent, different models can be compared, e.g., the similarity of a Petri net and an EPC can be calculated.

6 Related Work

The concept of a causal footprint based similarity measure relates to two research areas: (1) declarative approaches for process specification and (2) similarity of process models.

Causal graphs (or causal footprints) can be seen as a *declarative language*, i.e., instead of using explicit control-flow operators like sequence, iterations, etc., constraints are given. In this paper we consider two types of constraints (look-back links and look-ahead links) as introduced in [DMA06]. These can easily be translated into a temporal logic [MP91]. In [AP06] the Declarative Service Flow Language (DecSerFlow) is defined which includes the two types of constraints used in this paper but also many others. This illustrates that the approach presented in this paper could be extended to include other types of constraints in the causal footprint. In [AP06] it is shown how these can be represented in Linear

Temporal Logic (LTL) and a respective implementation of an LTL checker for event logs is included in the ProM framework [ABD05].

Existing work in the context of determining *similarity* between process models can be assigned to three categories: verification, integration, and degree of similarity. There are different notions of equivalence of process models that are subject to *verification* such as trace equivalence and bisimulation. While trace equivalence is based on a comparison of the sets of completed execution traces, bisimulation also considers at which point of time which decisions are taken, i.e., bisimulation is a stricter notion of equivalence. Details on different equivalence notions are given e.g. in [AAW06]. A general problem of such verification approaches is that it provides a true-false answer to the question whether two models are similar. The quantification of a degree of behavioral similarity between process models is an important contribution for the area of process model *integration*. While there are several approaches reported regarding *how* two models are integrated (see e.g. [PCS01, GRSS05, MS06]) the similarity degree gives an answer to the question *when* two process models might be a candidate for integration, e.g. in a merger situation.

Up to now, there is hardly any work reported on measuring the degree of *behavioral similarity* of two process models. In [AAW06] three similarity notions are discussed related to the structure, the state space, and the observed behavior of the models. The *structural similarity* counts connections between tasks that appear in both models and relates it either to the number of connections in the first benchmark model (called recall metric) or to those of the second model (called precision metric). Yet, it does not consider transitive order relations of tasks. The *naive behavioral similarity* is based on firing sequences of both models, but bears several problems with respect to termination, moment of choice, or finiteness of the state spaces. Instead, the authors propose a *behavioral similarity* metric based on the fitness of a set of event logs. This is indeed valuable for process mining, but not directly applicable for model comparison where no logs are present, or logs cannot be directly mapped to the higher-level description of the process (i.e. if there are transaction logs on one side, and a conceptual high level model on the other). In contrast to this work, causal footprint provide a rich set of information for comparison without the need to calculate the state space.

Finally, there are some approaches discussed for calculating the similarity of process models based on metadata (see e.g. [KB04, MS04, EKO07]). In contrast to the degree of similarity based on causal footprints, these contributions hardly use information about the behavior of the process models. Nevertheless, it seems possible to combine some of the ideas presented in these paper with causal footprints, e.g., semantical mappings between the labels of activities in different models.

7 Conclusion and Future work

In this paper, we presented a metric for comparing process models in different formats. We capture the *intent* of a process model by deriving a *causal footprint*. Such a footprint should be seen as an abstraction of the process behavior. As a causal footprint does

not require an executable semantics of the process modeling language, we can apply our analysis technique both to formal languages such as Petri nets and to conceptual/informal languages such as EPCs. Even though we use only these two languages throughout the paper to demonstrate the applicability of our technique, it can be easily adapted to other languages such as UML activity diagrams, BPMN, or BPEL. This is especially helpful regarding the heterogeneity of business process modeling languages. Furthermore, the causal footprints provide a rich set of information over models that can be utilized to determine the degree of similarity between them.

The degree of behavioral similarity based on causal footprints provides the basis for a more efficient querying and management of large collections of business process models, even if they are represented in different modeling languages. In particular the reengineering of such model collections would profit from our concepts since a similarity matrix might clearly indicate which pairs of process models are likely candidates for integration or deletion.

References

[Aal98] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[AAW06] W.M.P. van der Aalst, A.K. Alves de Medeiros, and A.J.M.M. Weijters. Process Equivalence: Comparing Two Process Models Based on Observed Behavior. In S. Dustdar, J.L. Faideiro, and A. Sheth, editors, *International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 129–144. Springer-Verlag, Berlin, 2006.

[ABD05] W.M.P. van der Aalst, H.T. de Beer, and B.F. van Dongen. Process Mining and Verification of Properties: An Approach based on Temporal Logic. In R. Meersman and Z. Tari et al., editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005*, volume 3760 of *Lecture Notes in Computer Science*, pages 130–147. Springer-Verlag, Berlin, 2005.

[ADK02] W.M.P. van der Aalst, J. Desel, and E. Kindler. On the Semantics of EPCs: A Vicious Circle. In M. Nüttgens and F.J. Rump, editors, *Proceedings of the EPK 2002: Business Process Management using EPCs*, pages 71–80, Trier, Germany, November 2002. Gesellschaft für Informatik, Bonn.

[ADMV06] W.M.P. van der Aalst, B.F. van Dongen, J. Mendling, and H.M.W. Verbeek. Fouten in SAP-referentiemodel (in Dutch). Automatisering Gids, 19th May 2006.

[AHKB03] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[AP06] W.M.P. van der Aalst and M. Pesic. DecSerFlow: Towards a Truly Declarative Service Flow Language. In M. Bravetti, M. Nunez, and G. Zavattaro, editors, *International Conference on Web Services and Formal Methods (WS-FM 2006)*, volume 4184, pages 1–23, 2006.

[BYRN99] R.A. Baeza-Yates and B.A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[CKL97] T. Curran, G. Keller, and A. Ladd. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Enterprise Resource Planning Series. Prentice Hall PTR, Upper Saddle River, 1997.

[DAV+05] B.F. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A new era in process mining tool support. In *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.

[DE95] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.

[DJVVA] B.F. van Dongen, M.H. Jansen-Vullers, H.M.W. Verbeek, and W.M.P. van der Aalst. Methods for EPC Verification and application to the Aris for MySAP reference models. *Computers in Industry (accepted for publication)*.

[DMA06] B.F. van Dongen, J. Mendling, and W.M.P. van der Aalst. Structural Patterns for Soundness of Business Process Models. In *EDOC '06: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pages 116–128, Washington, DC, USA, 2006. IEEE Computer Society.

[EJL+99] H. Eertink, W. Janssen, P. Oude Luttighuis, W. B. Teeuw, and C. A. Vissers. A business process design language. In *FM '99: Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems-Volume I*, pages 76–95, London, UK, 1999. Springer-Verlag.

[EKO07] M. Ehrig, A. Koschmider, and A. Oberweis. Measuring similarity between semantic business process models. In J.F. Roddick and A. Hinze, editors, *Conceptual Modelling 2007, Proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM 2007)*, volume 67, pages 71–80, Ballarat, Victoria, Australia, 2007. Australian Computer Science Communications.

[GRSS05] G. Grossmann, Y. Ren, M. Schrefl, and M. Stumptner. Behavior based integration of composite business processes. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Business Process Management, 3rd International Conference, BPM 2005, Nancy, France, September 5-8, 2005, Proceedings*, volume 3649 of *Lecture Notes in Computer Science*, pages 186–204. Springer, 2005.

[KB04] M. Klein and A. Bernstein. Toward high-precision service retrieval. *IEEE Internet Computing*, 8(1):30–36, 2004.

[Kin06] E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. *Data and Knowledge Engineering*, 56(1):23–40, 2006.

[KNS92] G. Keller, M. Nüttgens, and A.W. Scheer. Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken, 1992.

[KT98] G. Keller and T. Teufel. *SAP(R) R/3 Process Oriented Implementation: Iterative Process Prototyping*. Addison-Wesley, 1998.

[MADV06a] J. Mendling, W.M.P. van der Aalst, B.F. van Dongen, and H.M.W. Verbeek. Errors in the SAP Reference Model. BPTrends, June 2006.

[MADV06b] J. Mendling, W.M.P. van der Aalst, B.F. van Dongen, and H.M.W. Verbeek. Referenzmodell: Sand im Getriebe - Webfehler. *iX - Magazin für Professionelle Informationstechnik. (in German)*, pages 131–133, August 2006.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.

[MS04] M. Momotko and K. Subieta. Process query language: A way to make workflow processes more flexible. In G. Gottlob, A.A. Benczúr, and J. Demetrovics, editors, *Advances in Databases and Information Systems, 8th East European Conference, ADBIS 2004, Budapest, Hungary, September 22-25, 2004, Proceesing*, volume 3255 of *Lecture Notes in Computer Science*, pages 306–321. Springer, 2004.

[MS06] J. Mendling and C. Simon. Business Process Design by View Integration. In Johann Eder and Schahram Dustdar, editors, *Proceedings of BPM Workshops 2006*, volume 4103 of *Lecture Notes in Computer Science*, pages 55–64, Vienna, Austria, 2006. Springer-Verlag.

[Mur89] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[MVD+07] J. Mendling, H.M.W. Verbeek, B.F. van Dongen, W.M.P. van der Aalst, and G. Neumann. Detection and Prediction of Errors in EPCs of the SAP Reference Model. *Data and Knowledge Engineering (accepted)*, 2007.

[PCS01] G. Preuner, S. Conrad, and M. Schrefl. View integration of behavior in object-oriented databases. *Data & Knowledge Engineering*, 36(2):153–183, 2001.

[RR98] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.

[SWY75] G. Salton, A. Wong, and C.S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, 1975.

Forderungen an hierarchische EPK-Schemata

Volker Gruhn, Ralf Laue
{gruhn,laue}@ebus.informatik.uni-leipzig.de
Lehrstuhl für Angewandte Telematik und E-Business*
Universität Leipzig, Fakultät für Informatik

Abstract: Ereignisgesteuerte Prozessketten (EPK) sind eine Notation zur Modellierung von Geschäftsprozessen. Komplexe Geschäftsprozesse werden üblicherweise nicht als eine einzige EPK modelliert, sondern durch mehrere EPK-Schemata, die durch Prozesswegweiser und hierarchisierte Funktionen miteinander verbunden sind. In diesem Beitrag wird untersucht, welche Forderungen an solche EPK-Schemata zu stellen sind, um eine eindeutige Interpretation zu gewährleisten.

1 Einführung

Ereignisgesteuerte Prozessketten (EPK) sind eine Notation zur Modellierung von Geschäftsprozessen. Da sie im SAP-Referenzmodell angewendet wurden und zentraler Bestandteil der ARIS-Plattform sind, fanden sie insbesondere im deutschsprachigen Raum eine weite Verbreitung.

Ursprünglich wurden EPKs eingeführt, ohne deren Syntax und Semantik formell zu definieren[KNS92]. Zahlreiche Autoren schlugen später formale Definitionen zur Syntax und Semantik vor, um diese Lücke zu schließen. Eine (bei weitem nicht vollständige) Auswahl von einschlägigen Arbeiten ist [CS94, LSW97b, van99, NR02, Kin04, Men07].

Alle Arbeiten zur Definition einer Syntax und Semantik richten ihr Augenmerk allerdings kaum auf die Konstrukte, die dazu dienen, mehrere EPKs miteinander zu verbinden: Prozesswegweiser und hierarchisierte Funktionen. Diese Konstrukte sind in der Praxis sinnvoll, um zum einen eine bessere Übersichtlichkeit in der Modellierung zu erhalten, zum anderen Modellteile mehrfach wiederverwenden zu können.

[NR02] und [Rum99] formulieren grundlegende Anforderungen an Prozesswegweiser und hierarchisierte Funktionen, die den Ausgangspunkt für unsere weitergehenden Untersuchungen bilden.

In allen uns bekannten Veröffentlichungen wird davon ausgegangen, dass bei Einhaltung der durch [NR02] und [Rum99] aufgestellten Forderungen durch hierarchisierte Funktionen und Prozesswegweiser miteinander verbundene EPKs problemlos zu einer einzigen

*Der Lehrstuhl für Angewandte Telematik und E-Business ist ein Stiftungslehrstuhl der Deutschen Telekom AG

EPK zusammengefasst („flachgeklopft“) werden können. Wir zeigen in diesem Beitrag, dass dies keineswegs in jedem Falle ohne Schwierigkeiten möglich ist. Wir zeigen weiterhin einige Ansätze, um die Probleme zu lösen. Hauptanliegen dieses Beitrags soll es jedoch sein, anhand von Beispielen zu verdeutlichen, welche Schwierigkeiten die Aufgabe des „Flachklopfens“ mehrerer EPK-Schemata zu einem einzelnen flachen EPK-Schema aufwirft.

2 Grundlegende Definitionen

Für die weiteren Ausführungen setzen wir das Konzept der EPK als bekannt voraus. Bei der Formalisierung der EPK-Syntax und -Semantik orientieren wir uns an den in [NR02] und [Rum99] eingeführten Bezeichnungen. Wir vereinfachen diese jedoch auf diejenigen Elemente, die für unsere weiteren Betrachtungen relevant sind.

Definition 1 *(flaches EPK-Schema):*
Ein flaches EPK-Schema A ist ein Tupel $A = (E, F, P, V, J)$. Dabei ist

- *E eine nichtleere Menge von Ereignissen*
- *F eine nichtleere Menge von Funktionen*
- *P eine Menge von Prozesswegweisern*
- *V eine Menge von Verknüpfern (Konnektoren)*
- *$K = E \cup F \cup P \cup V$ die Menge aller Notationselemente*
- *$J \subset K \times K$ eine Menge von Kontrollflusskanten.*

E, F, P und V sind paarweise disjunkt.

Wir wollen im Weiteren ein flaches EPK-Schema kurz als EPK bezeichnen. An die Elemente eines flachen EPK-Schemas werden zusätzliche Anforderungen (etwa zur Multiplizität) gestellt, die in [NR02] und [Rum99] zu finden sind.

Zusätzlich werden die folgenden Bezeichnungen eingeführt:

- Für zwei Notationselemente $j, k \in K$ schreiben wir $j \rightarrow k$ genau dann, wenn $(j, k) \in J$, wenn es also eine Kontrollflusskante von j nach k gibt.
- $\rightarrow^*$ bezeichnet die transitive Hülle der durch $\rightarrow$ beschriebenen Relation.
- Die Menge der **direkten Vorgänger** eines Notationselements $k \in K$ ist $\bullet k = \{x \in K : (x, k) \in J\}$.
- Die Menge der **direkten Nachfolger** eines Notationselements $k \in K$ ist $k\bullet = \{x \in K : (k, x) \in J\}$.

- Für Notationselemente $j \in F \cup P$ definieren wir die Menge $VE(j)$ der **vorangehenden Ereignisse** als die Menge aller $e \in E$, für die $e \rightarrow j$ gilt oder es eine Folge von Verknüpfern $v_1, \ldots, v_n \in V$ gibt so dass gilt: $e \rightarrow v_1 \rightarrow \cdots \rightarrow v_n \rightarrow j$. Analog definieren wir die Menge $NE(j)$ der **nachfolgenden Ereignisse** als die Menge aller $e \in E$, für die $j \rightarrow e$ gilt oder es eine Folge von Verknüpfern $v_1, \ldots, v_n \in V$ gibt so dass gilt: $j \rightarrow v_1 \rightarrow \cdots \rightarrow v_n \rightarrow e$.
- Die Menge end(A) aller Endereignisse des flachen EPK-Schemas $A = (E, F, P, V, J)$ ist die Menge aller $e \in E$, für die kein $e' \in E$ mit $e \rightarrow^* e'$ existiert.
- Die Menge start(A) aller Startereignisse des flachen EPK-Schemas $A = (E, F, P, V, J)$ ist die Menge aller $e \in E$, für die kein $e' \in E$ mit $e' \rightarrow^* e$ existiert.[1]
- Ein $p \in P$ heißt **Startprozesswegweiser**, wenn $\bullet p = \emptyset$ und **Endprozesswegweiser**, wenn $p\bullet = \emptyset$.
- Ein Konnektor x heißt Split, wenn $\bullet x$ genau ein und $x\bullet$ mindestens zwei Elemente hat und Join, wenn $x\bullet$ genau ein und $\bullet x$ mindestens zwei Elemente hat.

3 Prozesswegweiser und hierarchisierte Funktionen

Mehrere flache EPK-Schemata können durch Prozesswegweiser und hierarchisierte Funktionen miteinander verbunden werden. Informell können wir die Bedeutung dieser Notationselemente wie folgt beschreiben:

Prozesswegweiser: Ein Prozesswegweiser leitet den Kontrollfluss von einer EPK zu einer anderen weiter. Es wird also „von einer EPK zu einer anderen gesprungen“, was mit einer GOTO-Anweisung in einer imperativen Programmiersprache vergleichbar ist.

Hierarchisierte Funktion: Trifft der Kontrollfluss in einer EPK auf eine hierarchisierte Funktion, so referenziert diese eine andere EPK. Der Kontrollfluss verzweigt zu der referenzierten EPK und kehrt zur „rufenden“ EPK zurück, wenn der Ablauf in der referenzierten EPK beendet ist. Dies ist vergleichbar mit einem Unterprogrammaufruf in einer imperativen Programmiersprache.

Wir führen wir die Menge der hierarchisierten Funktionen einer EPK in der folgenden Definition ein:

Definition 2 *(Hierarchisierte Funktion): Für eine EPK $A = (E, F, P, V, J)$ bildet die Menge $\widehat{F}$ der hierarchisierten Funktionen eine Untermenge der Funktionsmenge F.*

Formal werden Prozesswegweiser und hierarchisierte Funktionen in [NR02] und [Rum99] mit Hilfe einer Hierarchierelation beschrieben. Um die Schreibweise ein wenig zu vereinfachen, verwenden wir (ähnlich wie auch in [Men07] eingeführt) statt dessen den Begriff

[1] Die in manchen Quellen anzutreffende Definition von Start- und Endereignissen als Menge aller Ereignisse mit $e\bullet = \emptyset$ bzw. $\bullet e = \emptyset$ kann nicht verwendet werden, wenn Prozesswegweiser zugelassen sind.

der **Verweisfunktion**, die jedem Prozesswegweiser und jeder hierarchisierten Funktion das flache EPK-Schema zuweist, auf das sie verweisen.

Definition 3 *(Verweisfunktion, referenziertes EPK-Schema, EPK-Schemamenge, Verfeinerung):*
Sei $S = \{(E_1, F_1, P_1, V_1, J_1), \ldots, (E_n, F_n, P_n, V_n, J_n)\}$ *eine Menge flacher EPK-Schemata.*

Die Verweisfunktion h ist eine Funktion, die allen Prozesswegweisern, die in einem der EPK-Schemata aus S vorkommen, sowie allen hierarchisierten Funktionen, die in einem der EPK-Schemata aus E vorkommen, ein EPK-Schema aus S zuweist. Dieses zugewiesene EPK-Schema nennen wir im Folgenden auch „referenziertes Schema"; für ein $f \in \widehat{F}_i$ *heißt die EPK h(f) Verfeinerung von f.*

Somit ist die Abbildung h also erklärt als

$h : \bigcup_{i=1}^{n} \widehat{F}_i \cup P_i \longrightarrow S$

S selbst bezeichnen wir dann als EPK-Schemamenge.

Zur Bedeutung der Verweisfunktion ist zu beachten: Bei hierarchisierten Funktionen $f \in \widehat{F}$ ist h(f) die Verfeinerung von f. Für einen Endprozesswegweiser p_e ist $h(p_e)$ die EPK, zu der im Kontrollfluss „gesprungen" wird. Für einen Startprozesswegweiser p_s jedoch ist $h(p_s)$ die **„rufende"** EPK.[2] Wollen wir nur die „rufenden" Verweise betrachten, müssen wir die Startprozesswegweiser aus dem Definitionsbereich herausnehmen. Dies führt zu:

Definition 4 *(Aufruffunktion): Für ein EPK-Schema* $A_i \in S$ *bezeichne* P_i^{end} *die Menge der Endprozesswegweiser in* A_i*. Die Aufruffunktion h' ist die auf die Menge der Endprozesswegweiser und hierarchisierten Funktionen eingeschränkte Verweisfunktion h.*
h' ist also eine Abbildung $h' : \bigcup_{i=1}^{n} \widehat{F}_i \cup P_i^{end} \longrightarrow S$

4 Einfache Syntaxanforderungen an EPK-Schemamengen

[NR02] formuliert nun einige Anforderungen an EPK-Schemamengen wie folgt:

$E = \{A_1, \ldots, A_n\}$ sei eine EPK-Schemamenge mit der Verweisfunktion h. Dann muss gelten:

F 1 Die Menge der vorangehenden Ereignisse einer hierarchisierten Funktion entspricht der Menge der Startereignisse des referenzierten EPK-Schemas. Es gilt also:
$\forall f \in \widehat{F} : VE(f) = start(h(f))$

F 2 Die Menge der nachfolgenden Ereignisse einer hierarchisierten Funktion entspricht der Menge der Endereignisse des referenzierten EPK-Schemas. Es gilt also:

[2] In [NR02] und [Rum99] wird dieser Unterschied nicht beachtet, was dort zu Inkorrektheiten in den Definitionen führt.

$\forall f \in \widehat{F} : NE(f) = end(h(f))$

F 3 Die Menge der vorangehenden Ereignisse eines Endprozesswegweisers ist eine Teilmenge der Startereignisse des referenzierten EPK-Schemas. Es gilt also:

$\forall p \in P : p\bullet = \emptyset \Rightarrow VE(p) \subseteq start(h(p))$

F 4 Die Menge der nachfolgenden Ereignisse eines Startprozesswegweisers ist eine Teilmenge der Endereignisse des referenzierten Schemas. Es gilt also:

$\forall p \in P : \bullet p = \emptyset \Rightarrow NE(p) \subseteq end(h(p))$

F 5 Verbot der Rekursion: Ein EPK-Schema $A \in S$ ist nicht über mehrfache Anwendung der Aufruffunktion h' mit sich selbst verbunden. Es gibt also keine Folge $A_0, \ldots A_n \in S$, so dass für jedes $i = 0, \ldots, n-1$ ein Element $x_i \in \widehat{F}_i \cup P_i^{end}$ existiert, so dass $h'(x_i) = A_{i+1}$ und $A_0 = A_n = A$ gilt.

Man beachte bei den Punkten 3 und 4, dass die vorangehenden Ereignisse eines Endprozesswegweisers in der „aufrufenden", die nachfolgenden Ereignisse eines Startprozesswegweisers in der „gerufenen" EPK liegen.

In den folgenden Abschnitten werden wir Unzulänglichkeiten dieser Definitionen benennen und Verbesserungen vorschlagen.

5 Verbesserte Forderungen zur Aufrufhierarchie

Das Verbot der Rekursion in F5 soll vermeiden, dass sich eine EPK unendlich oft selbst aufrufen kann. Allerdings erscheint uns das Verbot eines solchen Selbstaufrufs im Falle von Prozesswegweisern als zu strikt. Gegen Modelle wie in Abb. 1 (zu finden z.B. im Referenzmodell „Ergebnisplanung" in [Kel99]) ist wenig einzuwenden. Der Verweis eines Endprozesswegweisers auf einen Startprozesswegweiser der selben EPK dient hier nur der Übersichtlichkeit und ersetzt lediglich einen Kontrollflusspfeil. In Abb. 1 ist dies sicher wenig sinnvoll, bei größeren Modellen kann es aber die Übersichtlichkeit erhöhen. Tatsächlich rekursive Aufrufe finden beim Verfolgen von Prozesswegweisern nicht statt.

Zu fordern ist lediglich, dass es in den Zyklen von EPKs, die sich per Prozesswegweiser einander aufrufen, in mindestens einer EPK Start- bzw. Endereignisse gibt, damit die Bearbeitung auch tatsächlich begonnen bzw. beendet werden kann.

Kritisch dagegen ist eine „Nacheinanderschaltung" von Aufrufen hierarchisierter Funktionen und dem Verfolgen von Prozesswegweisern zu sehen. Abb. 2 zeigt einen Ausschnitt aus einer EPK, die eine hierarchisierte Funktion aufruft. Will man mit dieser EPK arbeiten, muss man sich darauf verlassen können, dass nach Eintreten des Ereignisses „Vorbedingung" die hierarchisierte Funktion H ausgeführt wird und anschließend immer das Ereignis „Nachbedingung" eintritt. Eine genaue Betrachtung der durch H referenzierten EPK h(H), also ein Ändern der Abstraktionsebene, darf für das Verständnis der EPK nicht erforderlich sein.

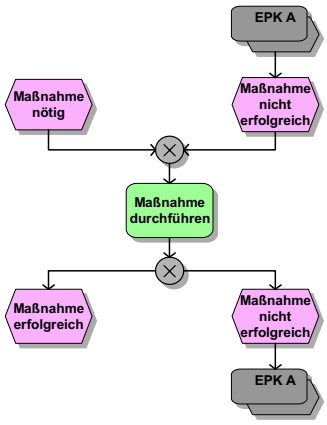


Abbildung 1: Rekursiver Prozesswegweiser

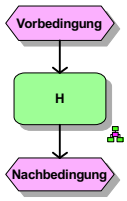


Abbildung 2: Aufruf einer hierarchisierten Funktion

Enthielte aber nun die referenzierte EPK h(H) ausgehende Prozesswegweiser, die auf eine andere EPK als h(H) selbst verweisen, könnte dies dazu führen, dass der Kontrollfluss derart verzweigt, dass das Ereignis „Nachbedingung“ möglicherweise nie eintritt und die Kontrolle nicht an die übergeordneten EPK zurückgegeben wird. Besonders schlimm ist dabei, dass man einen solchen möglichen Ablauf nicht durch Betrachten der übergeordneten EPK erkennen kann. Ähnliche bei Betrachtung der übergeordneten EPK unerwartete Abläufe sind denkbar, wenn die von einer hierarchisierten Funktion referenzierte EPK eingehende Prozesswegweiser enthält. Um solchen Problemen zu begegnen, kann man die Forderung aufstellen, dass eine EPK, die Verfeinerung einer hierarchisierten Funktion ist, keine heraus- oder hereinführenden Prozesswegweiser haben darf. [3] Uns ist bewusst, dass diese Forderung eine recht starke Einschränkung ist, die durchaus der heute teilweise üblichen Praxis widerspricht.

Zusammenfassend ergeben sich die folgenden Forderungen, die Punkt 5 aus Abschnitt 4 ersetzen:

[3] Darüberhinaus sollte eine EPK, die Verfeinerung einer hierarchisierten Funktion ist, sogar stark sound[van99] sein, um Effekte ähnlich der beschriebenen zu vermeiden. Dies ist im Übrigen ein gutes Argument gegen schwächere Soundnesskriterien[DR01]: EPKs, die nicht stark sound sind, eignen sich nicht als Verfeinerung einer hierarchisierten Funktion. Diese Betrachtungen sind aber schon semantischer Natur und führen daher über das Anliegen dieses Beitrags hinaus.

F 5' Verbot der Rekursion über hierarchisierte Funktionen: Ein EPK-Schema $A \in S$ ist nicht über mehrfache Anwendung der Aufruffunktion h' auf hierarchisierte Funktionen mit sich selbst verbunden. Es gibt also keine Folge $A_0, \ldots A_n \in S$, so dass für jedes $i = 0, \ldots, n-1$ eine Funktion $x_i \in \widehat{F_i}$ existiert, für die $h'(x_i) = A_{i+1}$ und weiterhin $A_0 = A_n = A$ gilt.

F 6 Gibt es eine Folge von EPK-Schemata $A_0, \ldots A_n \in S$ mit $A_0 = A_n$, so dass für jedes $i = 0, \ldots, n-1$ ein Prozesswegweiser p_i in A_i enthalten ist, für den $h'(p_i) = A_{i+1}$, so muss mindestens ein EPK-Schema der genannten Folge ein Startereignis s mit $\bullet s = \emptyset$ und mindestens ein EPK-Schema der genannten Folge ein Endereignis e mit $e\bullet = \emptyset$ enthalten.

F 7 Verbot von ein- und abgehenden Prozesswegweisern in EPK-Schemata, die eine hierarchisierte Funktion verfeinern: Ist eine EPK A Verfeinerung einer hierarchisierten Funktion, so darf A keine Prozesswegweiser enthalten, die auf eine andere EPK als A selbst referenzieren.

6 Korrespondenz zwischen End- und Startprozesswegweisern

In den in Abschnitt 4 genannten Forderungen fehlt bisher noch eine naheliegende Bedingung, nämlich dass es zu jedem rufenden Prozesswegweiser einen gerufenen gibt und umgekehrt.

Man könnte daher folgende Forderung formulieren:

Seien A_1 und A_2 EPK-Schemata, die zu einer Schemamenge gehören. Gibt es in A_1 einen Endprozesswegweiser e mit $h(e) = A_2$, so muss es in A_2 einen Startprozesswegweiser s mit $h(s) = A_1$ geben. Existiert umgekehrt in A_2 ein Startprozesswegweiser s mit $h(s) = A_1$, so muss es in A_1 einen Endprozesswegweiser e mit $h(e) = A_2$ geben.

Aber auch mit dieser Forderung ist noch nicht garantiert, dass Prozesswegweiser aus einem rufendem und einem gerufenem EPK-Schema zusammenpassen. Die drei in Abb. 3 gezeigten Modelle erfüllen sämtliche bisher genannten Forderungen, jedoch stimmen die Ereignisse an den aus- und abgehenden Prozesswegweisern nicht überein, was ein Zusammenfügen der drei Modelle zu einem Gesamtmodell unmöglich macht.

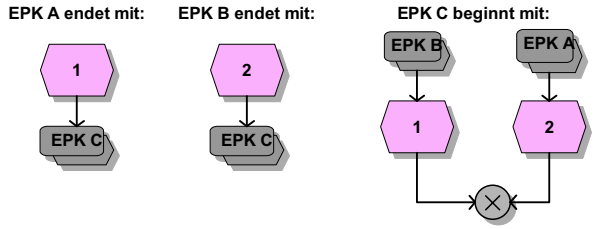


Abbildung 3: Ereignisse an rufender und gerufener EPK stimmen nicht überein

Bei dem Versuch, allgemeine Regeln aufzustellen, wann Ereignisse an Start- und Endprozesswegweisern „zusammenpassen“, trifft man auf die Schwierigkeit, dass sich in der Praxis im Einklang mit den bisher betrachteten syntaktischen Forderungen verschiedene Varianten der Benutzung von Prozesswegweisern durchgesetzt haben.
Abb. 4 zeigt zwei Möglichkeiten, von einer EPK per Prozesswegweiser auf eine andere zu verweisen. Auf den ersten Blick mag man vermuten, dass beide Varianten gleichwertig sind. Dies ist jedoch nicht der Fall. Der Unterschied besteht darin, dass der AND-Kontrollblock im Fall b) bereits abgeschlossen ist. Wie ist das Modell zu interpretieren, wenn die gerufene EPK die in Abb. 5 gezeigte Form hat? Im Falle a) lassen sich rufende und gerufene EPK leicht zu einem Modell zusammenfügen: Sie werden an den doppelt vorkommenden Ereignissen A und B verbunden. Zwar kann man dem erhaltenen Modell wegen des Aussprungs aus dem XOR-Kontrollblock schlechten Modellierungsstil vorwerfen, es ist jedoch durchaus akzeptabel.

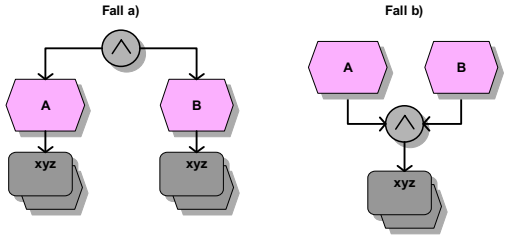


Abbildung 4: Verschiedene Arten der Benutzung von Prozesswegweisern

Anders ist die Situation im Falle b). In der aufrufenden EPK wird davon ausgegangen, dass beide Ereignisse A und B schon eingetreten sind, ehe zur aufgerufenen EPK gesprungen wird. Die gerufene EPK modelliert jedoch einen Ablauf, bei dem die Funktion F durchaus eintreten darf, wenn zwar B, aber noch nicht A eingetreten ist. Folglich können die beiden Modelle nicht sinnvoll zusammengefügt werden.

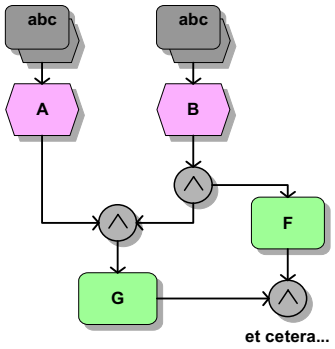


Abbildung 5: Gerufene EPK

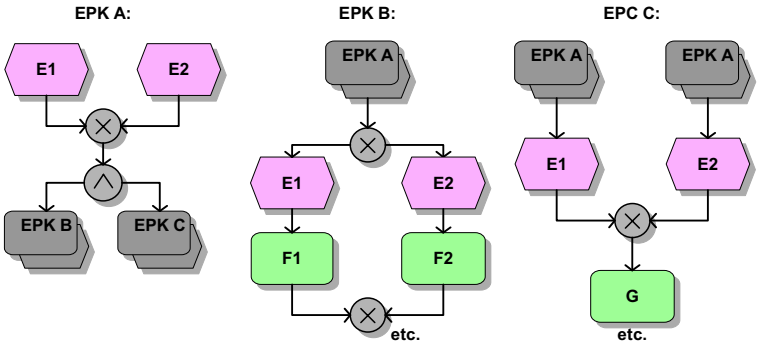


Abbildung 6: Problematische Konstruktionen, kein „Flachklopfen“ möglich

Einen ähnlichen Fall zeigt Abb. 6. Einzeln betrachtet, kann man sowohl EPK A mit EPK B als auch EPK A mit EPK C verbinden. Probleme bereitet aber das Zusammenfügen aller drei Modelle, da die Frage „Darf F1 gleichzeitig mit G ausgeführt werden?“ nicht befriedigend beantwortet werden kann: Die Kontrolllogik von EPK B verlangt, dass F1 und F2 *vor* dem XOR-Join in die EPK A eingefügt werden. F1 könnte demnach nicht gleichzeitig mit G ausgeführt werden. Das widerspricht aber der Kontrollflusslogik in EPK A, wonach F1 und F2 offenbar erst *nach* dem AND-Split einzufügen sind und somit F1 und G durchaus gleichzeitig ausgeführt werden können. Ein sinnvolles Zusammenfügen der drei EPKs ist unmöglich.

Abb. 7 zeigt schließlich den Aufruf einer hierarchisierte Funktion. Das Modell der Verfeinerung besagt, dass die Ausführung der Funktion „Planung“ bewirkt, dass die Ereignisse 1 und 2 eintreten. Das aufrufende Modell jedoch besagt etwas anderes, nämlich dass die Ereignisse 1 und 2 erst eintreten dürfen, wenn auch die Funktion „Bestellung“ ausgeführt wurde. Wegen dieses offensichtlichen Widerspruchs ist ein Zusammenfassen der beiden Modelle nicht möglich.

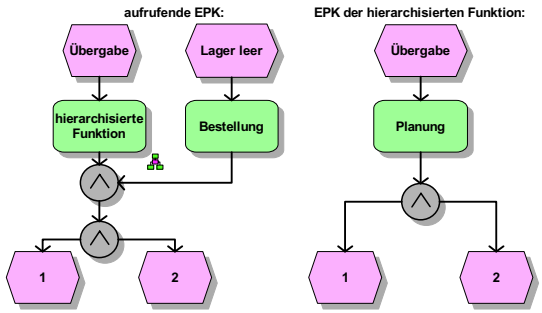


Abbildung 7: Problematische Konstruktionen, kein „Flachklopfen“ möglich

Zur Vermeidung der gezeigten Problemen führen wir im kommenden Abschnitt Beschränkungen für die Verwendung von Prozesswegweisern und hierarchisierten Funktionen ein und zeigen, wie unter diesen Restriktionen ein Flachklopfen erfolgen kann.

7 Restriktionen der Schnittstellen an Prozesswegweisern und hierarchisierten Funktionen

Wir definieren zunächst zwei Typen von Prozesswegweisern wie folgt:

Definition 5 *Ein Startprozesswegweiser s, für den s● ein Ereignis ist sowie ein Endprozesswegweiser e, für den ●e ein Ereignis ist, heißt Prozesswegweiser vom Typ 1. Ein Startprozesswegweiser s, für den s● ein Konnektor ist sowie ein Endprozesswegweiser e, für den ●e ein Konnektor ist, heißt Prozesswegweiser vom Typ 2.*

In Abb. 4 zeigt Fall a) zwei Prozesswegweiser vom Typ 1, während Fall b) einen Prozesswegweiser vom Typ 2 zeigt.

Durch zusätzliche Forderungen wollen wir nun sicherstellen, dass End- und Anfangsprozesswegweiser zusammenpassen (vgl. Abb. 8).

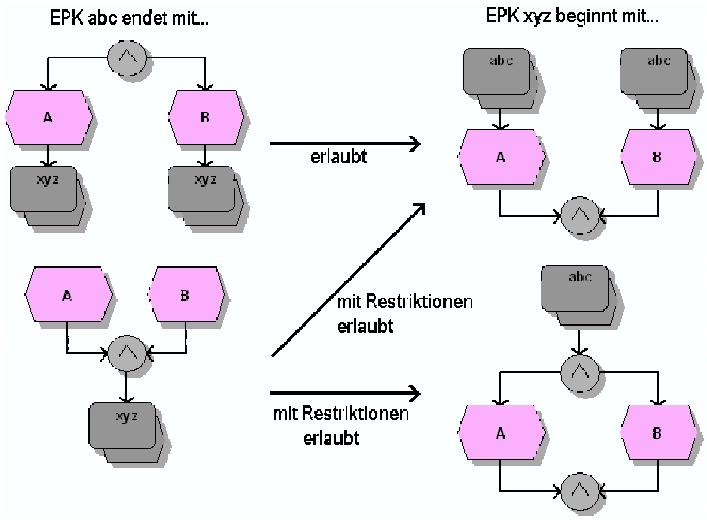


Abbildung 8: Restriktionen bei Aufrufen zwischen Prozesswegweisern

7.1 Endprozesswegweiser vom Typ 1

Prozesswegweiser vom Typ 1 stellen den einfachereren Fall dar. Jedem Endprozesswegweiser geht direkt ein Ereignis voran und jedem Startprozesswegweiser folgt direkt ein Ereignis. Um sicherzustellen, dass zwei EPKs zusammenpassen (vgl. Abb. 3 als Beispiel, wo das nicht der Fall ist), fordern wir:

F 8 Gibt es in einer EPK A_1 einen Endprozesswegweiser p_1 vom Typ 1 mit $h(p_1) = A_2$, so muss in der EPK A_2 ein Startprozesswegweiser p_2 vom Typ 1 mit $h(p_2) = A_1$ existieren, so dass $\bullet p_1 = p_2 \bullet$.

F 9 Gibt es in einer EPK A_1 einen Startprozesswegweiser p_1 vom Typ 1 mit $h(p_1) = A_2$, so muss in der EPK A_2 ein Endprozesswegweiser p_2 vom Typ 1 mit $h(p_2) = A_1$ existieren, so dass $p_1 \bullet = \bullet p_2$.

Das Zusammenfassen („Flachklopfen“) der beiden EPKs ist in diesem Fall problemlos: Die Prozesswegweiser werden entfernt und die EPKs an den in beiden EPKs vorkommenden Ereignissen zusammengefügt.

7.2 Endprozesswegweiser vom Typ 2

Während bei Prozesswegweisern vom Typ 1 rufende und gerufene EPK nur gemeinsame Ereignisse haben, besitzen sie bei Prozesswegweisern vom Typ 2 gemeinsame Konstrukte aus Ereignissen und Konnektoren. Man kann das so interpretieren, dass in der gerufenen EPK auf den Prozesswegweiser zunächst ein Block aus Ereignissen und Konnektoren folgt, der die in der rufenden EPK genannten Vorbedingungen für das Verfolgen des Prozesswegweisers wiederholt. Wie in Abb. 5 zu sehen ist, kann es Probleme geben, wenn in diesem Block bereits etwas von der eigentlichen Ausführungslogik der gerufenen EPK enthalten ist.

Um solche Fälle auszuschließen, fordern wir, dass in der gerufenen EPK dem Startprozesswegweiser ein Teilgraph folgt, dessen einzige Aufgabe es ist, die „Übergabelogik“ zu wiederholen, die durch die dem Prozesswegweiser der rufenden EPK vorausgehenden Ereignisse und Konnektoren vorgegeben ist.

Was wir hierbei unter „Übergabelogik“ verstehen, soll zunächst anhand von Abb. 9 erläutert werden. Hier gelangt der Prozesswegweiser genau dann zur Ausführung, wenn Ereignis 3 sowie entweder Ereignis 1 oder Ereignis 2 eintritt. Die logische Formel (1 xor 2) $\wedge$ 3, die diesen Sachverhalt beschreibt, nennen wir Übergabelogik von P_1.

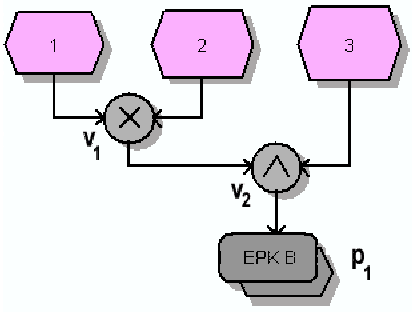


Abbildung 9: Übergabelogik (1 xor 2) $\wedge$ 3

Formal bestimmen wir die Formel, die die Übergabelogik an einem Endprozesswegweiser bestimmt, mit folgendem Algorithmus:

Zur Initialisierung enthalte die Zeichenkette Z den betroffenen Endprozesswegweiser, die Mengen S und J seinen leer. Solange dann die Zeichenkette Z noch EPK-Elemente enthält, die keine Ereignisse sind, führe folgenden Algorithmus aus:

1. Bestimme $\bullet x$ für alle in Z enthaltenen EPK-Elemente x, die keine Ereignisse sind.
2. Ist $\bullet x$ ein Split, so ersetze in Z x durch $\bullet x$ und füge $\bullet x$ der Menge S hinzu.
3. Ist $\bullet x$ ein Join vom Typ $\oplus$ (d.h. $\oplus \in \{xor, \vee, \wedge\}$) so ersetze in Z x durch $(x_1 \oplus \cdots \oplus x_n)$, wobei $\bullet x = \{x_1, \ldots, x_n\}$ die Menge der Vorgänger des Joins ist, und füge $\bullet x$ zur Menge J hinzu.

Ist der Algorithmus abgeschlossen, enthält Z als Zeichenkette eine logische Formel, die wir die Übergabelogik des Endprozesswegweisers nennen.

Außerdem werten wir noch die Mengen S und J aus. Gibt es ein $(s, j) \in S \times J$ mit $s \rightarrow^* j$, wird Z als unbestimmt betrachtet. Bei Endprozesswegweisern mit unbestimmter Übergabelogik ist zu vermuten, dass sie ein falsch modelliertes oder zumindest zweifelhaftes Konstrukt abschließen, so dass wir solche Endprozesswegweiser ausschließen wollen. Abb. 10 zeigt ein Beispiel für ein solches Konstrukt. Auch wenn das gezeigte Modellfragment theoretisch völig korrekt ist, spricht einiges dafür, solche Konstrukte zu verbieten. Weniger restriktive Einschränkungen zu finden, um manche „gutartigen“ Modelle mit unbestimmter Übergabelogik doch nicht auszuschließen, wäre eine Aufgabe für weiterführende Untersuchungen.

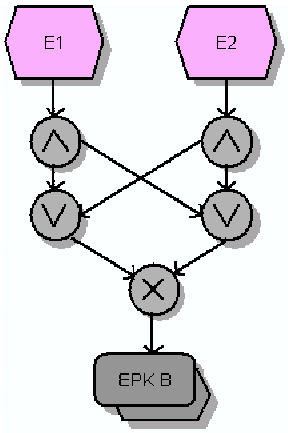


Abbildung 10: Endprozesswegweiser mit unbestimmter Übergabelogik

Für die in Abb. 9 gezeigte EPK sieht die Abarbeitung des Algorithmus wie folgt aus: Zunächst initialisieren wir Z durch $Z = P_1$, wobei P_1 den Prozesswegweiser bezeichnet. Im nächsten Schritt wird $\bullet P_1 = V_2$ bestimmt. Da V_2 ein Join vom Typ $\wedge$ ist, dessen

Vorgänger der Konnektor V_1 und das Ereignis 3 sind, setzen wir im nächsten Schritt $Z = (V_1 \wedge 3)$. Da V_1 ein Join vom Typ xor ist und dessen Vorgänger die Ereignisse 1 und 2 sind, lautet das Ergebnis des nächsten Ausführungsschrittes $Z = ((1 \text{ xor } 2) \wedge 3)$. Da alle in dieser Zeichenkette enthaltenen EPK-Elemente Ereignisse sind, kommt der Algorithmus zum Ende und liefert mit Z die Übergabelogik des Prozesswegweisers P_1.

Sei nun allgemein p ein Endprozesswegweiser und Z(p) seine Übergabelogik. Dann ist zu fordern, dass sich die durch Z(p) beschriebene Übergabelogik am Anfang der referenzierten EPK h(p) wiederfindet. Dazu fordern wir, dass h(p) eine Kante (x,y) enthält, die den Übergabeteil (der die Übergabelogik beschreibt) vom eigentlichen Ausführungsteil (der die Ablauflogik beschreibt) trennt. Für eine solche Kante gilt, dass der die EPK beschreibende Graph nach Entfernen der Kante nicht mehr zusammenhängend ist. (In der Graphentheorie wird hierfür die Bezeichnung cut-vertex verwendet.) Somit ist x das letzte Element des Übergabeteils und y das erste Element des Ausführungsteils.

Abb. 11 zeigt die obigen Bezeichnungen an einem Beispiel.

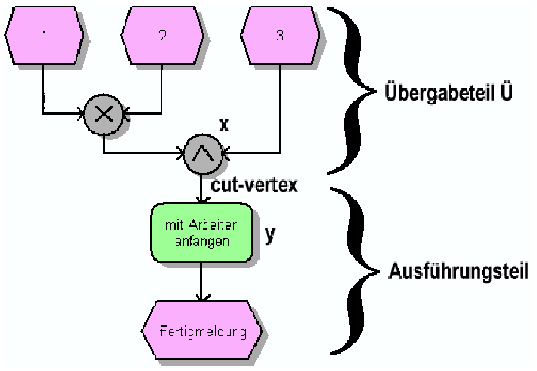


Abbildung 11: Cut Vertex trennt Übergabeteil von Ausführungsteil (Die Startprozesswegweiser wurden in dieser Abbildung weggelassen.)

Die Übergabelogik des Übergabeteils (Symbol: Z(h(p))) definieren wir wieder mit Hilfe des oben angeführten Algorithmus. Für die Anfangsinitialisierung wählen wir jetzt allerdings y, also das der cut-vertex folgende Element.

Um eine Korrespondenz zwischen ausgehendem Prozesswegweiser und Übergabeteil der referenzierten EPK zu gewährleisten, fordern wir dann, dass die logischen Formeln Z(p) und Z(h(p)) logisch identisch sind. Damit ist gewährleistet, dass die durch Ereignisse und Konnektoren beschriebenen Vorbedingungen für die Ausführung eines Prozesswegweisers mit den für die Abarbeitung der referenzierten EPK nötigen Bedingungen übereinstimmen. Es ist zu beachten, dass wir nicht die Gleichheit der Zeichenketten Z(p) und Z(h(P)) fordern. Es reicht die logische Identität. Für den in Abb. 9 beschriebenen Fall mit $Z(P_1) = (1 \text{ xor } 2) \wedge 3)$ wäre also z.B. durchaus ein Übergabeteil der referenzierten EPK möglich, der die Übergabelogik $Z(h(P_1)) = (1 \wedge 3) \text{ xor } (2 \wedge 3)$ besitzt.

Wir formulieren nun die folgende Forderung:

F 10 Hat eine EPK e_1 einen Endprozesswegweiser p_1 vom Typ 2, und sei $Z(p_1)$ die Übergabelogik von p_1. Dann muss die referenzierte EPK $e_2 = h(p_1)$ eine cut-vertex (x,y) mit der folgenden Eigenschaft besitzen:

Bezeichnen wir den Teilgraph von e_2, der x enthält, als Übergabeteil Ü, dann gilt:

Ü ist zyklenfrei und enthält keine Funktionen. Für die Übergabelogiken gilt, dass $Z(p_1)$ und Z(y) logisch identisch sind. Ferner trifft einer der beiden folgenden Fälle zu:

(a) Jedes Startereignis s in Ü ist hat als Vorgänger einen Startprozesswegweiser $\bullet s = p_i$ mit $h(p_i) = e_1$. Darüberhinaus enthält Ü keine Splits.

(b) In Ü gibt es genau einen Startprozesswegweiser vom Typ 1. Dieser referenziert auf e_1. Ferner ist Ü ein sauber geschachtelter Kontrollblock.[4]

Für die in Abb. 9 gezeigte EPK sind in Abb. 12 Beispiele für referenzierte EPKs zu sehen, die den angeführten Regeln entsprechen. Das „Flachklopfen“ besteht in beiden Fällen darin, in der gerufenen EPK den Übergabeteil Ü zu entfernen und das der cut-vertex folgende Element an Stelle des rufenden Prozesswegweisers zu setzen.

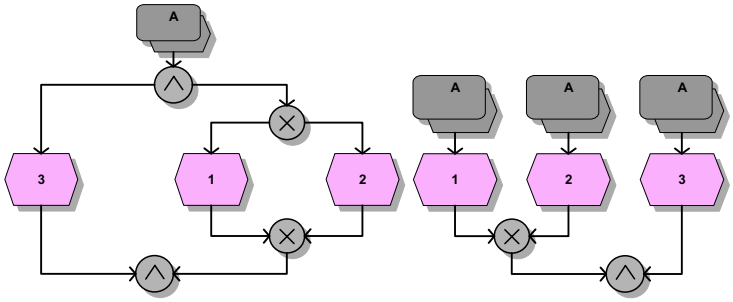


Abbildung 12: Passende EPK zu Abb. 9, links: Startprozesswegweiser haben Typ 1, rechts: Startprozesswegweiser haben Typ 2

7.3 Zusammenfassung der Prozesswegweiser-Forderungen

In den beiden vorangehenden Unterkapiteln haben wir für Endprozesswegweiser vom Typ 1 und vom Typ 2 Forderungen aufgestellt, denen die referenzierte EPK genügen soll. Wir haben ferner beschrieben, wie die EPKs an den Prozesswegweisern zusammengefügt werden können, wenn unsere Forderungen erfüllt sind.

Der Vollständigkeit halber erwähnen wir noch, dass natürlich auch zu fordern ist, dass es zu jeder EPK mit Startprozesswegweiser auch eine zugehörige rufende EPK gibt:

[4] „Sauber geschachtelt“ heißt hier: Splits und Joins treten paarweise auf und sind vom gleichen Typ, eine formale Definition findet sich z.B. in [LSW97a].

F 11 Enthält eine EPK e_1 einen Anfangsprozesswegweiser p mit $h(p) = e_2$, so muss e_2 einen Endprozesswegweiser q mit $h(q) = e_1$ enthalten.

Die von uns beschriebenen Restriktionen an die Verwendung von Prozesswegweisern stellen sicher, dass die zu verbindenden EPKs zu einer EPK zusammengefügt werden können. Unserer Erfahrung nach entsprechen die meisten in der Praxis anzutreffenden EPKs mit Prozesswegweisern den genannten Restriktionen. Es gibt jedoch durchaus Fälle, wo sinnvoll modellierte EPKs mit Prozesswegweisern unsere Restriktionen nicht erfüllen, z.B. sind Konstruktionen wie in Abb. 13 durch die von uns beschriebenen Fälle noch nicht abgedeckt. Es gibt also noch zahlreiche Möglichkeiten zur Erweiterung der in diesem Beitrag beschriebenen Überlegungen. Auf Grund der Komplexität selbst zunächst einfach scheinender Fälle haben wir zunächst darauf verzichtet, alle theoretisch nur denkbaren Konstruktionen zu erfassen.

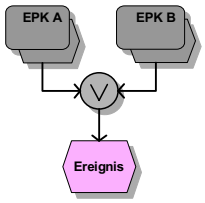


Abbildung 13: Noch nicht erfasster Fall

7.4 Hierarchisierte Funktionen

Nachdem wir im vorangegangenen Unterkapitel das Zusammenfügen von Prozesswegweisern untersucht haben, können wir nun auch das Flachklopfen hierarchisierter Funktionen mit Hilfe der dort beschriebenen Methode erklären.

Wie in Abb. 14 gezeigt, überführen wir den Aufruf einer hierarchisierten Funktion in einen Aufruf via Prozesswegweiser. Dazu setzen wir in der rufenden EPK einen Endprozesswegweiser an Stelle der hierarchisierten Funktion. In der referenzierten EPK setzen wir vor jedes Startereignis einen Startprozesswegweiser, der auf die rufende EPK referenziert.

Die Prüfung der Korrespondenz zwischen einer EPK mit hierarchisierter Funktion und der durch sie referenzierten Verfeinerung erfolgt dann wie zuvor für Prozesswegweiser beschrieben.

Analog lässt sich die Prüfung der Korrespondenz beim „Rücksprung“ von der Verfeinerung an die rufende EPK durchführen.

Die an Prozesswegweiser gestellten Anforderungen garantieren, dass für eine hierarchisierte Funktion f zwischen den Ereignissen in VE(f) und f keine Splits sowie zwischen f und den Ereignissen in NE(f) keine Joins vorkommen, was Situationen wie in Abb. 7 gezeigt vermeidet.

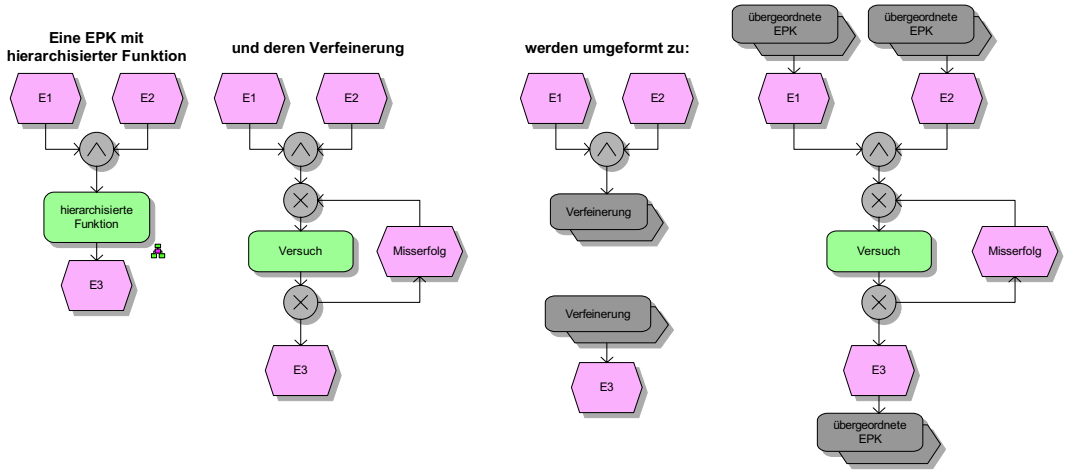


Abbildung 14: Ersetzten des Aufrufs einer Verfeinerung durch Prozesswegweiser

8 Weiterführende Fragestellungen

Wir haben bisher schon an einigen Beispielen gezeigt, dass es beim „Flachklopfen“ hierarchisierter EPKs verschiedene unschöne Situationen geben kann, die in der Literatur bisher nicht benannt wurden. Da das Thema komplexer ist, als man auf den ersten Blick vermutet, halten wir es für durchaus wahrscheinlich, dass noch weitere Schwierigkeiten gefunden werden.

Wie soll beispielsweise eine Situation behandelt werden, in der mehrere hierarchisierte Funktionen einer EPK auf die selbe Verfeinerung verweisen? Während es im linken Modell in Abb. 15 keine Probleme geben dürfte, resultiert das rechte Modell darin, dass die Verfeinerung mehrfach parallel auszuführen wäre, was in der Praxis sicher zu Ressourcenkonflikten führen würde. Ähnliche Fragestellungen sind für Prozesswegweiser denkbar. Hier wäre zunächst die Frage zu beantworten, welche Semantik ein mehrfaches Vorkommen einer Funktion oder eines Ereignisses in einer EPK haben soll. Obwohl solche Modelle in der Praxis nicht selten sind und z.B. auch durch das Austauschformat EPML[MN04] ausdrücklich unterstützt werden, ist diese Frage bisher nicht formal beantwortet.

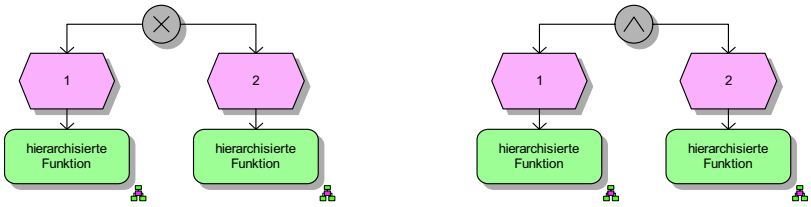


Abbildung 15: Mehrfaches Vorkommen einer Verfeinerung

Eine weitere wünschenswerte Forderung könnte es sein, dass man für jede EPK erkennen sollte, welche Startbelegungen gültig sind. Dies wäre z.B. für das in Abb. 16 gezeigte Modell nicht der Fall, da man hier erst an der Übergabelogik der referenzierten EPK 2 erkennen kann, ob die Ereignisse A und B gleichzeitig auftreten können bzw. müssen.

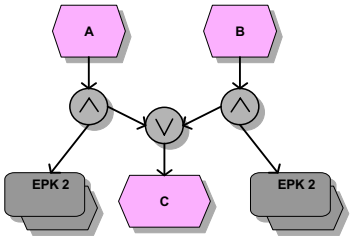


Abbildung 16: Welche Startbelegungen sind hier erlaubt?

9 Zusammenfassung

In unserem Beitrag haben wir syntaktische Anforderungen an EPKs mit hierarchisierten Funktionen und Prozesswegweisern präzisiert. Wir haben gezeigt, dass das „Flachklopfen“ mehrerer EPKs einer EPK-Schemamenge, das in der Literatur bisher immer als problemlos dargestellt wurde, zu verschiedenen unschönen Schwierigkeiten führen kann. Durch die Einführung von Restriktionen für hierarchisierte Funktionen und Prozesswegweiser beschrieben wir eine Klasse von EPKs, für die ein Flachklopfen automatisiert möglich ist. Allerdings stellen wir nicht die Behauptung auf, mit unseren Restriktionen den denkbar allgemeinsten Fall zu beschreiben, in dem ein Zusammenfassen von Modellen einer EPK-Schemamenge zu einem flachen EPK-Schema möglich ist. Die in diesem Beitrag beschriebenen Forderungen sind durchaus subjektiver Natur und sollen vor allem den Ausgangspunkt für weitere wissenschaftliche Diskussionen liefern.

Literatur

[CS94] R. Chen und A.W. Scheer. Modellierung von Prozessketten mittels Petri-Netz-Theorie. *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, (107), 1994.

[DR01] Juliane Dehnert und Peter Rittgen. Relaxed Soundness of Business Processes. In *CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, Seiten 157–170, London, UK, 2001. Springer-Verlag.

[Kel99] Gerhard Keller. *SAP R/3 prozessorientiert anwenden*. Addison-Wesley, München, 1999.

[Kin04] Ekkart Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In *Business Process Management*, Seiten 82–97, 2004.

[KNS92] G. Keller, M. Nüttgens und A.W. Scheer. Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, (89), 1992.

[LSW97a] P. Langner, C. Schneider und J. Wehler. Ereignisgesteuerte Prozessketten und Petri-Netze. *Berichte des Fachbereichs Informatik der Universität Hamburg*, (106), 1997.

[LSW97b] P. Langner, C. Schneider und J. Wehler. Prozessmodellierung mit ereignisgesteuerten Prozessketten (EPKs) und Petri-Netzen. *Wirtschaftsinformatik*, 39(5):479–489, 1997.

[Men07] Jan Mendling. *Detection and Prediction of Errors in EPC Business Process Models*. Dissertation, Vienna University of Economics and Business Administration, 2007.

[MN04] J. Mendling und M. Nüttgens. Exchanging EPC Business Process Models with EPML. In M. Nüttgens und J. Mendling, Hrsg., *XML4BPM 2004, Proceedings of the 1st GI Workshop XML4BPM – XML Interchange Formats for Business Process Management at 7th GI Conference Modellierung 2004, Marburg Germany, March 2004*, Seiten 61–80, March 2004.

[NR02] Markus Nüttgens und Frank J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In *Promise 2002 - Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen*, Seiten 64–77, 2002.

[Rum99] Frank J. Rump. *Geschäftsprozeßmanagement auf der Basis ereignisgesteuerter Prozeßketten*. B. G. Teubner Verlag Stuttgart Leipzig, 1999.

[van99] Wil M.P. van der Aalst. Formalization and verification of event-driven process chains. *Information & Software Technology*, 41(10):639–650, 1999.

Boolean and free-choice semantics of Event-driven Process Chains

Joachim Wehler
Ludwig-Maximilians-Universität München, Germany
joachim.wehler@gmx.net

Abstract. In parallel to the successful application of Event-driven Process Chains (EPCs) in commercial IT-projects there is an ongoing scientific debate about their semantics. It is even controversial, if one should provide EPCs with a local or with a non-local semantics. In this paper, we unify two local EPC semantics, which build on Petri nets. One of them is the free-choice semantics due to the translation of EPCs into free-choice systems. The other one is the Boolean semantics, which derives from translating the EPC into a certain class of high-level Petri nets. The free-choice semantics is restricted to EPCs with only AND or XOR-connectors, but allows arbitrary loops. The Boolean semantics allows connectors of arbitrary logical type. But at this stage, it covers only EPCs with well-structured loops. For EPCs, where both semantics are defined, we derive the free-choice semantics from the Boolean semantics and show as main result: The EPC is well-formed with respect to the Boolean semantics if and only if it is well-formed with respect to the free-choice semantics. After reviewing a series of examples from the literature we continue the investigation of the Boolean semantics for EPCs with circuits different from well-structured loops.

Keywords: EPC, local semantics, Boolean system, free-choice system, loop

1 Introduction

In the domain of business process management most commercial projects in Germany use the language of Event-driven Process Chains (EPCs) to model their processes. The widespread use of this language can be explained by two factors: In a successful manner EPCs combine intuitive comprehensibility with sufficient expressiveness. A second reason is a marketing aspect: EPCs are implemented into the tool ARIS[1], the market leader in the domain of business process modeling and controlling.
The language of EPCs has been invented by Keller, Nüttgens and Scheer in 1992 [KNS1992]. EPCs represent the control flow of a process as the interplay of three components: Events, functions and logical rules. The rules use connectors of logical type AND, XOR and OR. More specific, concurrency is represented by AND-splits and AND-joins. Strong or exclusive alternatives are modeled by XOR-splits and XOR-joins, while OR-splits and OR-joins model weak alternatives.

[1] ARIS is a product of IDS Scheer AG

Nearly from the beginning EPCs have attracted the interest of the academic domain. From 1994 until today there is a ongoing debate about the semantics of EPCs, about the concept of their well-formedness and about formal methods to verify EPCs.
The problem of the semantics for the logical EPC connectors shows up in full sharpness already for binary connectors. These are connectors with either one entry and two exits (*split*) or two entries and one exit (*join*). In the following we restrict to binary join- and split-connectors. About the semantics of AND-connectors there prevails agreement. Highly debated remains the semantics of XOR-connectors and OR-connectors. Most difficult seems the semantics of the OR-join.
The different approaches concerning the semantics of the connectors fall into two classes: *Local* semantics and *non-local* semantics. A local semantics considers only the given connector, a non-local semantics takes into consideration also a subset of all process runs. Typical non-local semantics are the semantics of Kindler [Kin2006], a typical local semantics is the free-choice semantics of van der Aalst [Aal1999].
The focus of this paper is to unify two different types of local EPC semantics. The paper is organized as follows: Chapter 2 *Examples of EPCs* presents a series of examples from the literature and provides them with a well-defined initial state. Chapter 3 *Non-local versus local semantics* shortly reviews some approaches to the problem of EPC semantics. Chapter 4 *Boolean semantics* is the main part of the paper. After unifying the free-choice and the Boolean semantics we prove: Well-formedness with respect to one semantics is equivalent to well-formedness with respect to the other semantics. The paper ends with Chapter 5 *Conclusion*, which also gives an outlook to further research.
The paper presupposes some familiarity with EPCs and Boolean systems. Such knowledge can be obtained by reading, e.g., [KNS1992], [Rum1999], [LSW1998], [Kin2006], ordered by increasing difficulty.

2 Examples of EPCs

In the present chapter we collect some EPCs from the literature. We will refer to them in later chapters to illustrate our reasoning about the semantics of EPCs.
The first question concerning the semantics of EPCs asks about the type of objects, which flow along the arcs of the EPC. In [KNS1992] the authors state, that an EPC represents the *flow of control*. Even when EPCs allow for augmenting each function by the processed data, these data do not participate in the flow.
All EPCs are assumed to be connected. EPCs from commercial projects are event-bounded, i.e. they start with a set of one or more events (*start* events) and end with one or more events (*end* events). The intended semantics expressed by this modeling rule is the following: The start events trigger the process, the end events describe the final state, which the process has to achieve. This simple rule assumes in particular, that a process has a final state at all, i.e. that it terminates.
Due to the lack of markings any EPC needs an additional specification: One has to determine, which combination of start events may happen and which combination of end events is intended. As long as the initial state of EPCs is undefined, one cannot expect a well-defined semantics for EPCs. Therefore several authors have borrowed the marking concept from formal process languages like Petri nets. Annotating the

EPC with a marking implies, that one considers no longer a plain EPC. Instead one asks for the semantics of a marked EPC.
In the present paper like in our previous paper [LSW1998] we use a different method, striving to stay within the range of the original EPC language. To provide an EPC with well-defined initial and final states we circuit each event-bounded EPC, i.e. we augment it with a *start/end connection*. A start/end connection introduces a separate event "start/end" and connects the events at the boundary to the new event with the help of the logical connectors from the EPC language. The initial state of an EPC occurs, when this distinguished event happens. The final state occurs, when it is achieved a second time. EPCs with start/end connection are strongly connected.
In the following we seek to present all examples as EPCs with start/end connection. Because we extracted some EPCs from the literature, we had to find out, which start/end connection expresses best the intention of the author. Sometimes it could be derived from an annotation of the EPC, for instance relating to the initial marking. But some other time it is unclear, if the EPC models a terminating process at all.
To simplify matters we often have omitted all events and functions, which are irrelevant to discuss the semantics of the logical connectors. Hence the EPCs are not necessarily shown with their correct syntax.

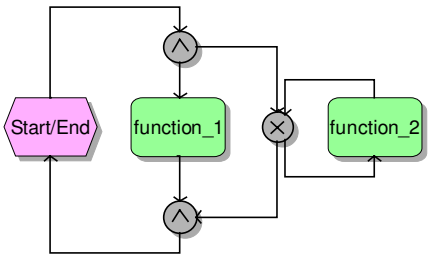


Fig. 1 Single loop

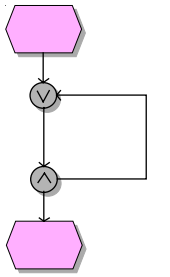


Fig. 2 Circuit [MA2006]

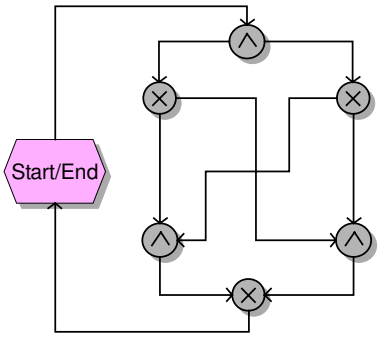


Fig. 3 Entangled AND and XOR [GT1984]

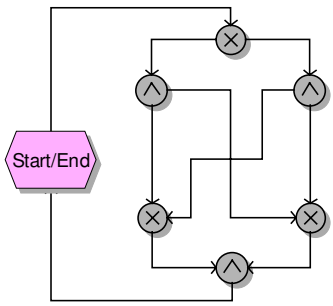


Fig. 4 Entangled XOR and AND [GT1984]

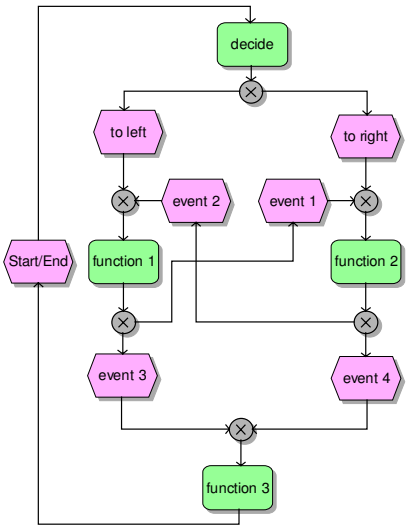


Fig. 5 Loop with multiple entries and exits

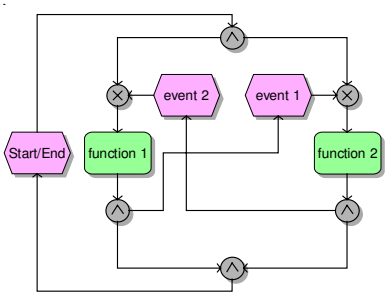


Fig. 6 Entangled circuit [Kin2006]

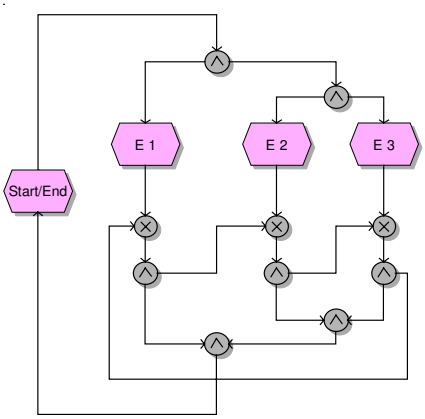


Fig. 7 Entangled circuit [Kin2006]

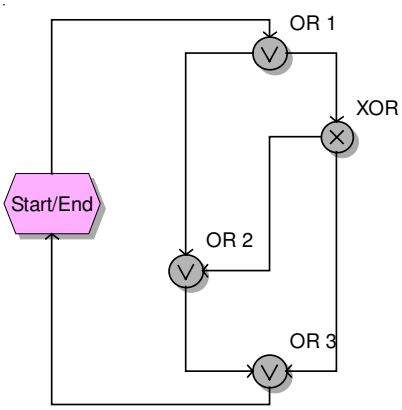


Fig. 8 Unpaired Or-connectors

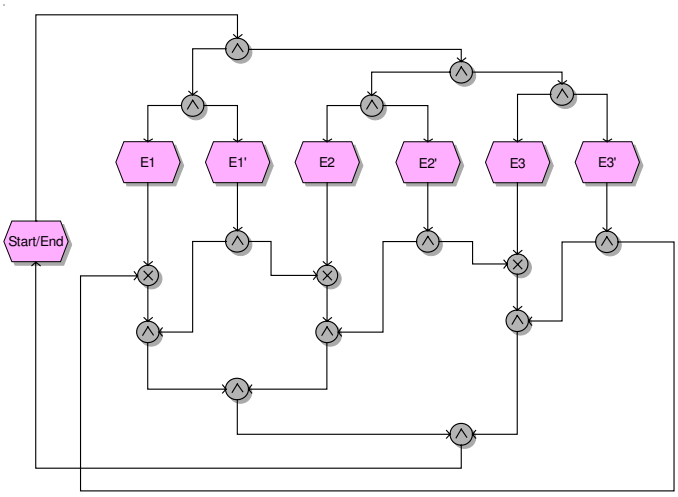


Fig. 9 Entangled AND/XOR [Kin2006]

3 Non-local versus local semantics

All authors reasoning about the semantics of EPCs have provided EPCs at least with the concept of state. Mostly, they even have translated EPCs into an other language with a well-defined semantics. Such languages are *transition systems* or *Petri nets*.

The most recent survey of proposals for EPC semantics is contained in [MA2006]. The authors also assess each proposal and state its limitations. Concerning non-local semantics the authors consider in particular the papers [Rit2000], [WEAH2005] and [Kin2006]. Concerning local semantics they refer to [CS1994][2], [LSW1998], [Aal1999] and [Rit2000]. In addition we would like to refer to [Rod1997] and [GL2005]. The work of Rodenhagen is an early proposal to translate EPCs into Petri nets. Gruhn-Laue translate EPCs into a modification of coloured Petri nets and use low- and high-tokens.

In our opinion the question of *non-local* semantics for EPCs has been settled by the result of Kindler. His work has been prompted by the attempt of Nüttgens-Rump [NR2002], to formalize their intended non-local semantics by a transition system. They introduced markings for the state of an EPC and considered transition systems as a formalization of statefull EPCs. The resulting transition system depends heavily on the semantics of the XOR-joins and OR-joins. In case one of the join-entries is marked, one has the choice between an optimistic non-local semantics and a pessimistic non-local semantics. In case of doubt the optimistic join fires, while the

[2] We consider the translation into coloured Petri nets a *local* EPC-semantics, because the constituents of Petri nets like places, arcs and transitions have a local semantics.

pessimistic one refrains from firing. The EPC has an *ideal semantics*, iff both non-local semantics coincide. Concerning the intended non-local semantics Kindler proves in [Kin2006]: An EPC has

1. either a unique ideal semantics
2. or two or more ideal semantics
3. or a pair of non-local semantics, which are not ideal but approximate best an intended ideal semantics.

An example from case 2 is the EPC in Fig. 6. Examples from case 3 are the cyclic EPCs in Fig. 7 and the acyclic EPC in Fig. 9. Apparently, in case 2 as well as in case 3 the EPC does not have a unique ideal semantics.

4 Boolean semantics

Now - the question of non-local semantics being settled – we focus on local EPC semantics. We do not aim at a new formal model for a semantics, which might have been intended by the creators of EPCs. Instead we unify two existing local semantics.
The Boolean semantics of an EPC is a local one: It is the semantics of a high-level Petri net, into which the EPC translates. A *state* of the EPC is a reachable marking of its Petri net.

Boolean system

A *Boolean system* is a coloured Petri net $BS = (BN, \mu)$ with a *Boolean net* BN and an *initial marking* μ. Each place of BN has the same token colour set

$$Boole = \{high, low\}.$$

A *high-token,* which marks a place, shows the presence of the control flow, while a *low-token* explicitly excludes the control flow. Every transition of a Boolean net has a binding colour set: An AND-transition has two binding elements, an XOR- transition has three and an OR- transition has four binding elements. But transitions of a Boolean net are not restricted to these logical types. More general formulas are allowed, specifically asymmetric OR-connectors like L_XOR are well-defined.

Logical type	Binding elements
AND	(1,1):=(high, high; high), (0,0):=(low, low; low)
XOR	(1,0):=(high, low; high), (0,1):=(low, high; high), (0,0):=(low, low; low)
OR	(1,0):=(high, low; high), (0,1):=(low, high; high) (1,1):=(high, high; high), (0,0):=(low, low; low)
L_XOR	(1,0):=(high, low; high), (1,1):=(high, high; high), (0,0):=(low, low; low)

Fig. 10 Binding elements of join-transitions of different logical type

Fig. 10 enumerates the binding elements or firing modes of join-transitions of different logical type. A notation like $(1,0) := (high, low, high)$ means that the binding

element *(1,0)* is enabled by a high-token at its left pre-place and a low-token at its right pre-place. When firing it creates a high-token at its post-place. The asymmetric L_XOR-join synchronizes the activation of both entries like an AND-join and it allows the exclusive activation of its left entry like one of the XOR-modes.
If one forgets about all colours of BS, i.e. about the difference between token elements and about the difference between all binding elements of the same transition, then one obtains the *skeleton* of the Boolean system: It is an ordinary Petri net

$$BS^{skel} = \left(BN^{skel}, \mu^{skel} \right)$$

with the same net structure, together with a canonical map

$$BS \longrightarrow BS^{skel}.$$

For the formal definition of Boolean nets and Boolean systems[3] we refer to ([LSW1998], Def. 2 and 7). All Boolean nets of the present paper are supposed to be *faithful concerning activation*: Any *high-mode*, i.e. any firing mode, which consumes at least one high-token, also creates at least one high-token.

As is well known, coloured Petri nets are no more than a compact notation for ordinary Petri nets. Every coloured Petri net expands into an ordinary Petri net, named its *flattening*: Every token element of a place of the coloured net induces a place of the flattening. Every binding element of the coloured net induces a transition of the flattening. And the token colour of any token of the coloured Petri net induces a token at the corresponding place of the flattening.
In case of a Boolean system $BS = \left(BN, \mu \right)$ the flattening of the different join-transitions from Fig. 10 are the ordinary nets from Fig. 11. An analogous flattening is obtained for the split-transitions just by reversing arcs. In Fig. 11 the gray places and transitions as well as their connecting arcs form part of an ordinary Petri net, which is called the *high-system*

$$BS^{high} = \left(BN^{high}, \mu^{high} \right)$$

of the given Boolean system. The white components form part of a second ordinary Petri net, which is called the *low-system* of the Boolean system. It is the subnet of the flattening, which is generated by all low-places and all low-transitions. Factoring out the low-system from the flattening gives the high-system as a quotient of the flattening.

[3] In the present paper it is not required as part of the definition, that the skeleton of a Boolean system is live and safe. But all examples will have this property

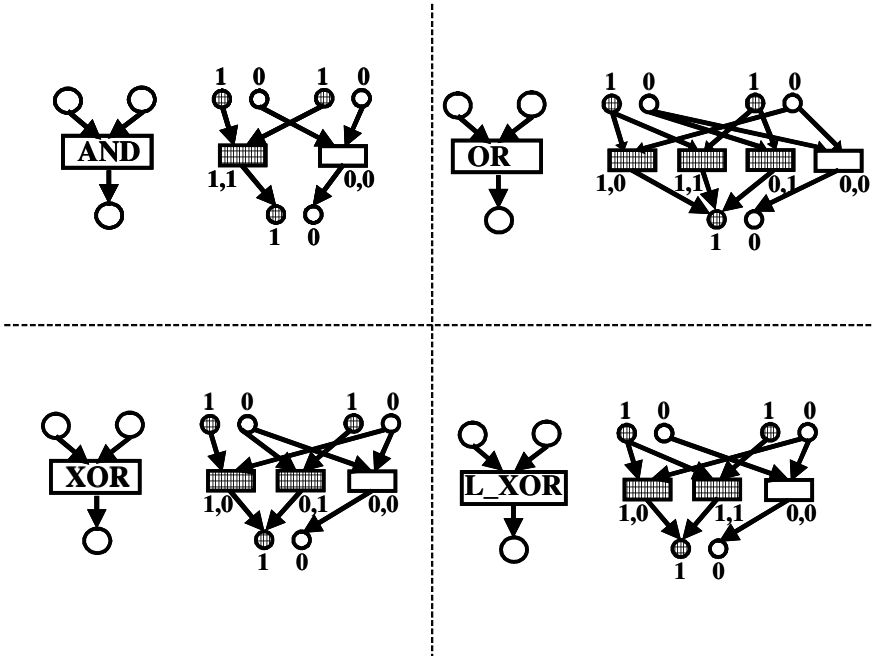


Fig. 11 Flattening of join-transitions of different logical type (1 = high, 0 = low)

If all transitions of a Boolean system are of logical type AND or XOR and if all places are unbranched, then the Boolean System is called a *bipolar system* ([LSW1998], Remark 7). Specifically, a bipolar system has no transitions of logical type OR. Bipolar systems have been invented by Genrich-Thiagarajan [GT1984].
The high-net of a bipolar system is a free-choice net of *restricted* type: Every place has either a single post-transition, or all post-transitions of the place have no other pre-place. For the definition of general free-choice systems and for their theory cf. [DE1995].

In the context of bipolar systems it is important to know, if the free-choice system excludes any *frozen tokens*. Absence of frozen tokens is a fairness property: It is not possible to fire an infinite occurrence sequence without moving all tokens.

1. Definition *(Frozen token)*
An ordinary Petri net (N,μ_0) has no *frozen tokens*, iff for every reachable marking μ holds: For every strictly less marking $\nu < \mu$ the Petri net (N,ν) has no infinite occurrence sequence.

2. Remark. For a live free-choice system the absence of frozen tokens is even a structural property: A live free-choice system has no frozen tokens iff it is safe and every T-component has a nonempty intersection with every P-component ([BD1990], Theor. 6.2).

In order to unify the Boolean and the free-choice semantics of an AND/XOR-EPC we use a result, which relates the behaviour of a bipolar system to the behaviour of its high-system and its skeleton. Here we characterize the behaviour of a Petri net by two properties named respectively safeness and liveliness.
A Petri net is *safe*, if any reachable marking marks no place with more than a single token. Safeness is an intuitive requirement for a Boolean system, which serves to model the control-flow of a system. Note, that safeness does not exclude the investigation of multiple process instantiation as long as different process instances do not interact.
We call a Boolean system *high-live*, if for any reachable marking, for any transition and for any of its high-modes there exists a follower marking, which activates the given high-mode. A high-live Boolean system does not contain any starving high-modes, which after a finite number of executions could not be enabled any longer. Liveliness is a plausible requirement to intensify the structural constraint of strong connectedness. It takes into account also the behaviour. Note that it is not reasonable to require that the Boolean system is live with respect to the low-modes, too.

3. Theorem *(Bipolar system and free-choice system)*
i) A bipolar system is high-live and safe if and only if its high-system is live and safe without frozen tokens and its skeleton is live and safe.
ii) Every restricted free-choice system, which is live and safe without frozen tokens, is the high-system of a bipolar system, which is high-live and safe.

Proof. Cf. [Weh2007], Theor. 4.6 and 4.9.

A free-choice net is named *well-formed*, if it has a live and bounded marking. Due to a result, which generalizes a theorem of Genrich, the existence of a live and bounded marking even implies the existence of a live and safe marking ([DE1995], Theor. 5.10). Hence a free-choice net is well-formed, iff it has a live and safe marking. For free-choice nets of restricted type, in particular for the high-nets of AND/XOR-EPCs, there exists a useful characterization of well-formedness due to Esparza-Silva [ES1990].

4. Theorem (*Well-formedness of restricted free-choice nets*)
A restricted free-choice net is well-formed, iff it is strongly connected, no elementary circuit has a TP-handle, and every PT-handle of an elementary circuit has a TP-bridge.

An *elementary path* in a net is a path without self-intersection. A *handle* at an elementary circuit is an elementary path, which starts and ends with a node of the circuit, but has no other node in common with the circuit. A handle, which starts with a transition and ends with a place, is called *TP-handle*. If it starts with a place and ends with a transition, it is called *PT-handle*. A *TP-bridge* of a handle at an elementary circuit is a path, which starts with a node of the handle and ends with a node of the circuit, but has no other nodes in common with the handle or the circuit. For formal definitions of these concepts cf. [ES1990].

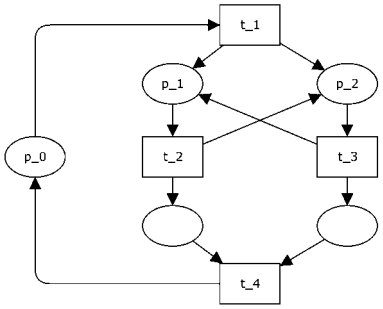


Fig. 12 Restricted free-choice net with TP-handles

The restricted free-choice net from Fig. 12 is not well-formed, because it contains several forbidden TP-handles: The elementary path (t_1, p_2, t_3, p_1) is a TP-handle at the left elementary circuit with six nodes, which passes through place p_0 and the three transitions t_1, t_2 and t_4.

Translation of EPCs without circuits

In this section we consider EPCs, which are a priori acyclic, i.e. they are circuit-free before performing their start/end connection from Chapter 2. Afterwards the circuited EPC translates into a Boolean system ([LSW1998], Proced. 1), which is an *elementary Boolean loop system*

$$EBLS = (BN, p, \mu):$$

The skeleton of the Boolean net BN is an *elementary loop* ([LSW1998], Def. 9). The place $p \in BN$ is the *baseplace*, i.e. the distinguished place corresponding to the event "start/end" of the EPC. The *initial marking* μ of BN has a single token, namely a high-token, which marks the baseplace. If $EBLS$ is high-live and safe, then the Boolean net BN and the original EPC is named *well-formed*.

The skeleton BN^{skel} as well as the high-net BN^{high} are strongly connected and the complements of their baseplaces p^{skel} and p^{high} do not contain any circuit. Note that all circuits from BN^{skel} and BN^{high} are elementary circuits, which pass through the baseplace. When cutting BN^{skel} or BN^{high} by splitting the baseplace into a source place and a sink place one obtains a Workflow net.

5. Remark (*Workflow net* [Aal1996])
A *Workflow net* WN is an ordinary net, which satisfies two conditions:

- It has two special places: a *source place* i with no ingoing arc, i.e. $pre(i) = \varnothing$, and a *sink place* o with no outgoing arc, i.e. $post(o) = \varnothing$, and

- after adding a new transition t with $pre(t) = o$ and $post(t) = i$ the net WN extends to a strongly connected net $\overline{WN}$.

By fusing the source and sink place of a Workflow net one obtains an ordinary net with baseplace, which is covered by circuits passing through the baseplace. Skeletons and high-nets of elementary Boolean loop systems are contained in the class of nets obtained from the fusion of Workflow nets. But the inclusion is proper, because elementary Boolean loop systems do not contain any loops, which omit the baseplace.

An a priori acyclic AND/XOR-EPC translates into a bipolar system. Its high-system is the same free-choice system, which has been detected by Genrich-Thiagarajan ([GT1984], Chap. 3). And it is exactly the free-choice system, which has been proposed by van der Aalst [Aal1999] as the translation of an AND/XOR-EPC.

6. Theorem *(Deriving the free-choice semantics from the Boolean semantics)*
The semantics of an a priori acyclic AND/XOR-EPC induced by the high-system of its Boolean system according to [LSW1998] equals the free-choice semantics according to [Aal1999].

Theorem 6 follows at once from the definition of the high-system. Accordingly the free-choice semantics of an a priori acyclic AND/XOR-EPC can be obtained from its Boolean semantics. But the Boolean semantics achieves more: It is defined for EPC connectors of arbitrary logical type. In particular, it provides the OR-join with a local semantics.

Note. In ([LSW1998], Def. 12) we proposed to model OR-connectors always as a completed alternative with a pair of split- and join-connectors. At this point we would like to emphasize, that this proposal is a modeling rule for EPCs, which facilitates their *analysis*. But the Boolean *semantics* of an OR-connector is well-defined also for isolated, unpaired connectors.

Van der Aalst discusses the OR-connector in the context of a completed OR-alternative ([Aal1999], Fig. 10). When marking both pre-places in Fig. 11 of transition (1,1) belonging to the OR-join, then not only transition (1,1), but also transition (1,0) and transition (0,1) are enabled. Van der Aalst rejects the gray subnet of the OR-join in Fig. 11 as a local semantics of the EPC connector. Instead he discusses three alternatives: First the resolution of an OR according to the formula

x OR y <=> (x AND y) XOR (x XOR y),

secondly the introduction of additional places equal to the two white low-places in Fig. 11 and thirdly a modification of the firing rule of Petri nets to a non-local rule. He rejects all three procedures. Note: Not only the second proposal but also the first one, which duplicates both branches of an OR-split, is not applicable for isolated OR-splits. In [Aal1999] the author abstains from defining a local semantics of the OR-connector. With different collaborators he devotes a series of successive papers to

the task to provide the OR-connector either with a non-local semantics, e.g., [WEAH2005] [4], or with a local semantics, e.g., [MA2006].
We follow the rejection of the high-net as a candidate for translating the OR-join. Fig. 11 shows the reason, why the high-net is insufficient to capture the intended semantics of an OR-join: The three transitions (1,1), (1,0) and (0,1) from the flattening of the Boolean OR-transition belong to a common cluster. Marking both pre-places from the low-net is necessary to decide which transition of the cluster is enabled. Apparently this cluster obstructs the free-choice property. In our opinion this fact prevents any translation of an OR-join into a free-choice net.

Analysis of behaviour

Safeness and liveliness are two properties to describe the behaviour of a Petri net. As a third property we now add being cyclic. A Boolean system is called *cyclic*, if the initial marking is reachable from any reachable marking. A cyclic Boolean system can always return to its initial state.

7. Lemma *(Liveliness and absence of frozen tokens)*
Consider an elementary Boolean loop system $EBLS = (BN, p, \mu)$.
i) The skeleton of $EBLS$ is a live and safe T-net.
ii) If $EBLS$ is high-live and safe, then it is also cyclic and any reachable marking, which marks the baseplace p, equals the initial marking μ.
iii) The high-system of a bipolar system $EBLS$ has no frozen tokens, if it is live and safe.

Proof. i) The skeleton is a strongly connected T-system. Each place belongs to a circuit and every circuit is marked at μ^{skel} with the single token at p^{skel}. Hence the skeleton is live and safe.
ii) We apply part i). It is well-known, that a live T-system is also cyclic. Moreover two reachable markings agree, if they mark every circuit with the same number of tokens ([DES1995], Theor. 3.21). Because every circuit passes through p^{skel}, the initial marking μ^{skel} is the only reachable marking with a token at p^{skel}. Now we lift this result from the skeleton to the Boolean system $EBLS$. By assumption $EBLS$ is high-live and therefore deadlock-free. Hence any enabled occurrence sequence of its skeleton lifts to an enabled occurrence sequence of $EBLS$. Consider a reachable marking μ_1 of $EBLS$. Any occurrence sequence of the skeleton, which is enabled at μ_1^{skel} and after firing creates the initial marking, lifts to an occurrence sequence of $EBLS$, which is enabled at μ_1 and after firing creates a marking μ_2 with a single token. This token marks the baseplace p. Because an elementary Boolean loop

[4] In Definition 16 of this paper there has to be added the requirement, that firing one of the resulting transitions in question consumes exactly one token from every marked pre-place.

system is faithful with respect to activation, this token must be a high-token, which proves $\mu = \mu_2$.
iii) Every elementary circuit of *BN* as well as of its high-net passes through respectively p and p^{high}. Because P-components and T-components of the high-net are strongly connected, they are covered by elementary circuits. Hence all P-components and all T-components of the high-net pass through p^{high}. According to Remark 2 the high-system has no frozen tokens, q. e. d.

Lemma 7, part ii) demonstrates a property, which has been called sound in the context of Workflow nets.

8. Remark (*Soundness* [Aal1996])
A Workflow net *WN* is named *sound*, if marking the source place with a single token provides an initial marking μ, such that (WN, μ) has the following properties:

- Each reachable marking has a follower marking, which marks the sink place,
- each reachable marking, which marks the sink place, does not contain a second token and
- for each transition t of *WN* there exists a reachable marking, which enables t.

It is well-known, that a Workflow net *WN* is sound, iff the extended net $\overline{WN}$ is live and bounded with respect to the induced marking $\overline{\mu}$.

Theorem 6 rises the following question concerning a priori acyclic AND/XOR-EPCs: How does soundness with respect to their free-choice semantics relate to soundness with respect to their Boolean semantics? The answer is given by Theorem 9, which heavily depends on Theorem 3:

9. Theorem *(Behaviour with respect to the Boolean and the free-choice semantics)*
Any a priori acyclic AND/XOR-EPC translates into a live and safe free-choice system, iff it translates into a high-live and safe Boolean system.

Proof. The elementary Boolean loop *EBLS* of the EPC is a bipolar system. Its high-system $EBLS^{high}$ defines the free-choice semantics of the EPC according to Theorem 6. If the bipolar system *EBLS* is high-live and safe, then its high-system $EBLS^{high}$ is live and safe according to Theorem 3. Conversely: According to Lemma 7, part i) the skeleton $EBLS^{skel}$ is live and safe. In order to apply Theorem 3 it suffices to show, that $EBLS^{high}$ has no frozen tokens. This follows from Lemma 7, part iii), q. e. d.

The two EPCs from Fig. 3 and Fig. 4 translate into elementary Boolean systems with the same skeleton, which is a live and safe T-system. But their corresponding high-systems differ. Both high-systems are safe. But the high-system of the EPC from Fig. 4 is also live, while the high-system of the EPC from Fig. 3 has a reachable marking, which is a deadlock. According to Lemma 7, part iii) and Theorem 3 the Boolean system of the EPC from Fig. 4 is high-live and safe, while the Boolean system of the

EPC from Fig. 3 is not high-live and safe. Hence the EPC from Fig. 4 is well-formed, while the EPC from Fig. 3 lacks well-formedness.
The AND/XOR-EPC from Fig. 9 translates into an elementary Boolean loop system, which is a bipolar system. Hence the Boolean semantics of the EPC is well-defined. The skeleton of the bipolar system is live and safe. The bipolar system is not high-live and safe according to Theorem 3. Because Theorem 4 shows, that the high-net is not well-formed due to the existence of TP-handles. The example is taken from ([Kin2006], Fig. 4). Kindler proves, that the EPC lacks a non-local ideal semantics.

EPCs with circuits

In the present section we consider a priori cyclic EPCs. They contain circuits already before attaching their start/end connection.
If a circuit of an EPC has a unique XOR-node, which serves as entry-point as well as exit-point, then the circuit is considered a loop and the intended semantics of the XOR-node is as following: The control flow either enters the loop or it passes the loop or it leaves the loop. Concerning entering or passing the node has the same semantics as an XOR-split: Either the control flow enters a given branch of the alternative or it passes this branch. After the control flow has entered the loop scope, it is decoupled from the rest of the net. There is no interaction between the control flow inside the loop and the control flow outside.
The free-choice semantics of an AND/XOR-EPCs considers only a single type of tokens which correspond to high-tokens. Hence it is suggesting that the free-choice semantics does not discriminate between an XOR-split at the start of a loop and an XOR-split at the start of an alternative. In both cases the XOR-splits translates into a place, branched in forward direction. A similar reasoning holds for XOR-joins.
The situation is different for the Boolean semantics. If the control flow enters a given branch at an XOR-split, it has decided not to activate the other branch. Hence the Boolean system propagates a high-token into the activated branch and a low-token along the other one. On the other hand: If the control flow enters the loop at an XOR-split, the decision about activation or not-activation of the loop's complement is postponed until the control flow exits from the loop. Hence the Boolean semantics propagates a high-token into the loop and no other token into its complement. Therefore the loop entry as well as the loop exit cannot be represented by a Boolean XOR-transition. Instead one has to use a branching place.
A Boolean system treats the high-token alike in both situations, but it discriminates concerning the low-token. Free-choice systems, which do not care about low-tokens, can deal with both situations in the same fashion.

In [LSW1998] we have undertaken a first step to deal with circuits of an EPC. We have investigated *well-structured loops*. A well-structured loop is an elementary circuit, which starts and ends with the same XOR-connector. The connector has two entries and two exits. EPCs with all elementary circuits of this type translate into a *Boolean loop system* $BLS = (BN, p, \mu)$: The Boolean net BN, being a *Boolean loop tree*, is covered by a finite family of elementary Boolean loops. Each elementary Boolean loop represents a single loop with its baseplace serving as loop-entry and as

loop-exit. Each loop is linked up in a tree-like fashion by fusing its baseplace with a distinguished place of the loop's complement ([LSW1998], Def. 9, 10). The distinguished place $p \in BN$ is the baseplace of the root and the marking μ of BN marks p with a single high-token and has no other tokens.

The *Boolean loop tree* as well as the original EPC is named *well-formed*, iff BLS is high-live and safe. Analogously to Lemma 7 one can prove the following result.

10. Remark *(Boolean loop system)*
Consider a Boolean loop system $BLS = (BN, p, \mu)$.
i) The skeleton of BLS is live and safe.
ii) A Boolean loop tree is well-formed, iff each of its elementary Boolean loops is well-formed.
iii) If BLS is high-live and safe, then it is also cyclic and any reachable marking, which marks the baseplace p, equals the initial marking μ.

Theorem 9 and Remark 10 imply the following Theorem 11.

11. Theorem *(Behaviour with respect to the Boolean and the free-choice semantics)*
Any AND/XOR-EPC, which has only well-structured loops, translates into a live and safe free-choice system, iff it translates into a high-live and safe Boolean loop system.

Thanks to Theorem 11 and 4 it is easy to verify the well-formedness of an AND/XOR-EPC with only well-structured loops: One checks the high-net of each elementary Boolean loop for the different types of handles and bridges. If the high-net is well-formed, then its basemarking is live and safe, because each P-component is marked with the single token at the basepoint (cf. [DE1995], Theor. 5.8 and 5.9).

The EPC from Fig. 1 with a single loop is well-structured. Fig. 13 shows its Boolean loop system BLS. It defines the Boolean semantics of the EPC. The Boolean loop tree is covered by two elementary loops, the root with baseplace p_0 and a second loop with baseplace p_1. Each elementary Boolean loop, when marking its baseplace with a high-token, is a high-live and safe elementary Boolean loop system.

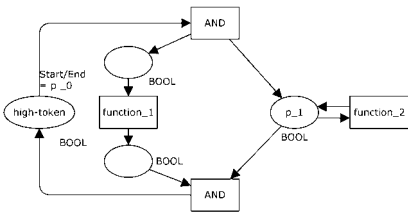


Fig. 13 Boolean loop system BLS

12. Remark *(Counterexample)*
Apparently one can disregard the loop character of the EPC from Fig. 1 and translate the AND/XOR-EPC directly into the bipolar system BS from Fig. 14. Its initial

marking has an additional low-token to assure liveliness of the skeleton. But *BS* does not provide the intended semantics of the given EPC. It is not high-live: Depending on the selected firing mode of the XOR-transition one can reach a deadlock.

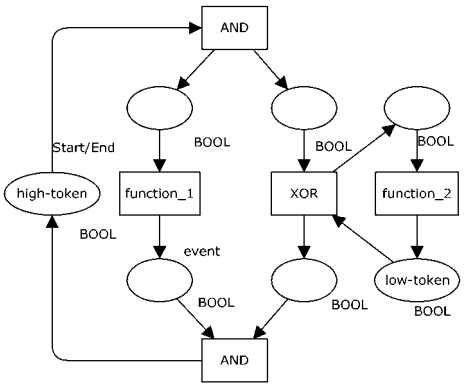


Fig. 14 Bipolar system *BS*

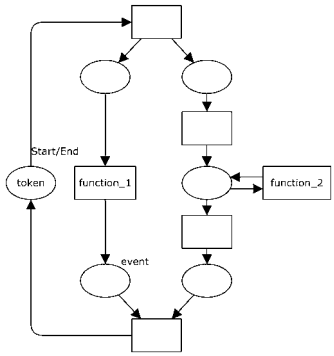


Fig. 15 High-system *FS* of *BS*

Fig. 15 shows the high-system *FS* of *BS* from Fig. 14. The free-choice system *FS* is live and safe, but it has a frozen token: After firing the split-transition one can fire the loop ("function_2") an infinite time without moving the token at the pre-place "event" of the join-transition. Hence Theorem 3 does not apply to this situation.
Note that *FS* defines the free-choice semantics of the EPC from Fig. 1 according to van Aalst. It also results from place fusing the high-systems of the two loop components of *BLS* from Fig. 13.
The EPC from Fig. 1 illustrates the following issue, mentioned in the beginning of this section: The Boolean semantics of an EPC discriminates between an XOR-connector either as entry point of a well-structured loop (Fig. 13) or as XOR-split of an alternative (Fig. 14). The free-choice semantics (Fig. 15) considers both the same.

Due to its specified composition from elementary loops a Boolean loop tree excludes any circuits as in Fig. 2, Fig. 5, Fig. 6 or Fig. 7. It does not provide a Boolean semantics for the corresponding EPCs. We do not even know, which process is intended by the EPC from Fig. 2. It seems reasonable to consider the event with the outgoing arc as start event. But what is the final state of the EPC, does there exist a final state at all, does the process terminate?

We now take a step forward in the investigation of EPCs with circuits. The three EPCs from Fig. 5, Fig. 6 and Fig. 7 have in common the existence of an elementary circuit, which neither passes through the baseplace nor forms a well-structured loop.

The EPC from Fig. 5 models the following scenario: You need an information, which can be obtained from two information offices. Therefore you decide, which office to ring up first. Assume you ring up the first office first ("function 1"). Either it answers the call or you ring up the second office ("function 2"). Now either the second office

answers the call or you try again the first office … . After leaving the loop you complete your work ("function 3").[5]

The elementary circuit passing through "function 1" and "function 2" is a loop with two entry points as well as with two exit points. We translate each XOR-connector, which corresponds to an entry or exit point, into a branching place of the Petri net. We have to assure, that no reachable marking marks the loop with more than one token. Therefore we connect all entry places with a single ENTER-connector to the places corresponding to the events "to_left" and "to_right". The connector waits for information, that one of the two events happens and the other cannot happen. Then its firing mode reduces the pair of high- and low-token at its two preplaces to a single high-token. This high-token initializes the loop at the entry point corresponding to the triggering event. The ENTER-connector has two high-modes and two low-modes according to Fig. 16. Note, that each firing mode consumes two tokens, but creates only one token.

Connector	Binding elements
ENTER	left_high := (high, low; high, -), right_high := (low, high; -, high), left_low := (low, low; low, -), right_low := (low, low; -, low)

Fig. 16 Binding elements of the ENTER-connector

Symmetrically, there is an EXIT-connector with analogous firing modes. It splits an arriving high-token into a pair of high- and low-tokens and an arriving low-token into a pair of low-tokens. For syntactical reasons we have to add inside the loop four unbranched transitions.

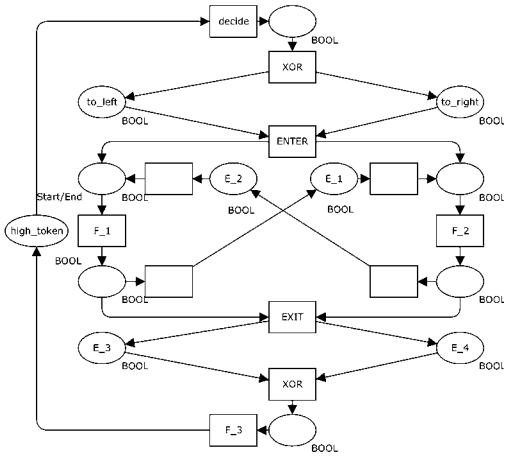


Fig. 17 BS as translation of the EPC from Fig. 5

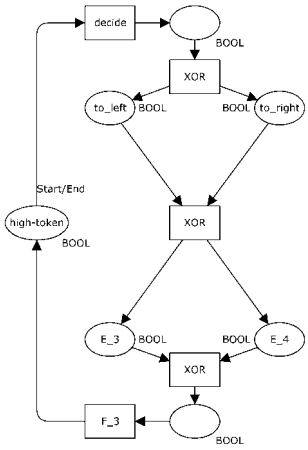


Fig. 18 Abstraction BS_{abstr}

[5] I thank Walter Vogler for communicating this example to me.

Fig. 17 shows the final Petri net BS. It is safe and every high-mode is live.

The EPC from Fig. 6 has been discovered by Kindler as an example of an EPC with two non-local, but ideal semantics. We refrain from translating the EPC into a Boolean system: The two exit points of the loop are represented by AND-connectors, but we do not know about a corresponding loop semantics.

The EPC from Fig. 7 has been discovered by Kindler as an example of an EPC without any ideal semantics at all. Instead, it has a pair of non-local semantics, which are best approximations of an intended ideal semantics. The loop has three exit points, which are represented by AND-connectors. By similar reasons we refrain from translating the EPC into a Boolean system.

The Boolean semantics of a hierarchy of EPCs

The Boolean semantics of an EPC is defined by a coloured Petri net. Hence the semantics of a hierarchy of EPCs is well-defined as soon as one has at hand an analogous concept for coloured Petri nets.
A hierarchy of coloured Petri nets is a family of coloured Petri nets, which are linked by morphisms in a tree-like fashion. We have proposed the concept of a morphism between coloured Petri nets in a previous paper ([Weh2007], Def. 2.6). The fibres of such a morphism refine a place by a place-bordered subnet, a transition by a transition-bordered subnet, a binding element by a T-flow and a token element by a P-flow. The morphism maps certain occurrence sequences of the Petri net in the domain to occurrence sequences of the Petri net in the image. Apparently the semantics of all Petri nets is independent from each other.

A simple example for such a morphism is the abstraction map

$$abstr : BS \longrightarrow BS_{abstr}$$

from the Petri net BS of Fig. 17 onto the Petri net BS_{abstr} of Fig. 18: The elementary circuit in the centre of BS as well as the transitions ENTER and EXIT map onto the central XOR-transition of BS_{abstr}. All other nodes and arcs from BS map injectively onto the corresponding net element of BS_{abstr}. Hence the EPC from Fig. 5, which translates into BS, abstracts onto the EPC corresponding to BS_{abstr}.

5 Conclusion

The task to formalize the intended semantics of EPCs is an ill-posed problem. As Kindler rightly remarks “Actually, there is not a single well-accepted informal semantics for EPCs. There are many variants and modifications.”

We are skeptic about all attempts to enrich EPCs with additional concepts like tokens, internal states, reset arcs, context, time … . These concepts are either introduced in an

ad-hoc fashion or they are borrowed from formal process languages, which have their own well-defined semantics. Of course such concepts are useful, the concept of state is even indispensable for process modeling. Instead of borrowing concepts from other languages, we are advocating to first translate the EPC into a target language and then to argue on the solid ground of that formal language.

Since 1994 we have accomplished numerous process modeling projects in the commercial field. Never there was a problem to read a given EPC as a Boolean net - at least after untangling the intended loops. And after discussing with the domain experts the intended start/end connection it was also possible to provide the initial marking. Of course this type of discussion with the customer from the user department seldom uses explicit formal reasoning. Neither it applies any algorithm, which transforms a given EPC into a well-formed one. Main ingredients are the consultants experience with similar cases and her knowledge about the standards and pitfalls of process modeling.
Questions concerning the intended semantics of EPCs are challenging from a theoretical point of view. But in the context of commercial applications they are not debated. We explain this lack of interest in a formal semantics with the imprecision of "real-world" EPCs. Up to now, commercial projects had no need to model precise EPCs, which are executable like a program.

In our opinion the question of *non-local* semantics for EPCs has been settled by the remarkable result of Kindler: There is no single transition relation, which captures the non-local semantics intended by Nüttgens and Rump. In general, there exist two related transition relations, which approximate best the intended semantics from two different directions.

The problem to handle EPC circuits in the context of Boolean systems is not yet solved. The example from Fig. 5 and the proposal of its translation in Fig. 17 mark the starting point of some work in progress. We study Boolean systems with components corresponding to loops with multiple entry and exit points.

Concerning Boolean systems another question seems interesting from a theoretical point of view. For a high-live and safe bipolar system all binding elements of the low-system are irrelevant for the flow of the high-tokens. The next step would be to study high-live and safe Boolean systems with transitions of logical type L_XOR or OR: Does there exist a maximal, non-empty subsystem of the low-system, which can be factored out without disturbing the flow of the high-tokens? One of the simplest Boolean systems in question is the Boolean system of the EPC from Fig. 8.

Acknowledgments

I thank the anonymous referees as well as Peter Rittgen for their helpful comments during the review. I tried to follow their suggestions and to answer their questions.

6 References

[Aal1996] *van der Aalst, Wil*: Structural Characterizations of Sound Workflow Nets. Computing Science Reports 96/23, Eindhoven University of Technology, Eindhoven, 1996

[Aal1999] *van der Aalst, Wil*: Formalization and verification of event-driven process chains. Information & Software Technology, 41 (10), 1999, p. 639-650

[BD1990] *Best, Eike; Desel, Jörg*: Partial Order Behaviour and Structure of Petri Nets. Formal Aspects of Computing (1990), p. 123-138

[CS1994] *Chen, R.; Scheer, August-Wilhelm*: Modellierung von Prozeßketten mittels Petri-Netz Theorie. Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 107, Saarbrücken 1994

[DE1995] *Desel, Jörg, Esparza, Javier*: Free Choice Petri Nets. Cambridge University Press, Cambridge 1995

[ES1990] *Esparza, Javier; Silva, Manuel*: Circuits, Handles, Bridges and Nets. In: *Rozenberg, Grzegorz* (Ed.) Advances in Petri nets 1990. Lecture Notes in Computer Science, vol. 483, Springer, Berlin 1990, p. 210-242

[GL2005] *Gruhn, Volker; Laue, Ralf*: Einfache EPK-Semantik durch praxistaugliche Stilregeln. In: *Nüttgens, Markus; Rump, Frank* (Hrsg.): EPK 2005. Geschäftsprozeßmanagement mit Ereignisgesteuerten Prozeßketten. Proceedings, Hamburg 2005

[GT1984] *Genrich, H.J.; Thiagarajan, P.S.*: A Theory of Bipolar Synchronization Schemes. Theoretical Computer Science 30 (1984), p. 241-318

[Kin2006] *Kindler, Ekkhart*: On the semantics of EPCs: Resolving the vicious circle. Data & Knowledge Engineering 56, (2006), p. 23-40

[KNS1992] *Keller, Gerhard; Nüttgens, Markus; Scheer, August-Wilhelm*: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)". Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, Saarbrücken 1992

[LSW1998] *Langner, Peter; Schneider, Christoph; Wehler, Joachim*: Petri Net based Certification of Event-driven Process Chains. In: *Desel, Jörg; Silva, Manuel* (Eds.): Application and Theory of Petri Nets 1998. Lecture notes in Computer science, vol. 1420. Springer, Berlin et al. 1998

[NR2002] *Nüttgens, Markus; Rump, Frank*: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: *Desel, Jörg; Wesel, M.* (Hrsg.): Promise 2002 – Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen. GI Lecture Notes in Informatics P-21, pp. 64-77, Gesellschaft für Informatik, 2002

[MA2006] *Mendling, Jan; van der Aalst, Wil*: Towards EPC Semantics based on State and Context. In: *Nüttgens, Marcus; Rump, Frank*: EPK 2006. Geschäftsprozeßmanagement mit Ereignisgesteuerten Prozessketten. 5. Workshop der Gesellschaft für Informatik, Wien, 2006

[Rit2000] *Rittgen, Peter*: Paving the road to Business Process Automation. In Proc. of the Europ. Conf. on Information Systems (ECIS), p. 313-319, 2000

[Rod1997] *Rodenhagen, Jörg*: Darstellung ereignisgesteuerter Prozeßketten (EPK) mit Hilfe von Petrinetzen. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Hamburg 1997

[Rum1999] *Rump, Frank*: Geschäftsprozeßmanagement auf der Basis ereignisgesteuerter Prozeßketten. Teuber, Stuttgart et al. 1999

[Weh2007] *Wehler, Joachim*: Free Choice Petri Nets without frozen tokens and Bipolar Synchronization Systems. arXiv: cs/0609095, 2007

[WEAH2005] *Wynn, Moe; Edmond, David; van der Aalst, Wil; ter Hofstede, Arthur*: Achieving a General, Formal and Decidable Approach to the OR-join in Workflow using Reset nets. In *Ciardo, G.; Darondeau, P.* (Eds.): Applications and Theory of Petri Nets 2005, vol. 3536 of Lecture Notes in Computer Science, p. 423-443. Springer, Berlin et al. 2005

Verarbeitung von ARIS-EPK-Modellen im Eclipse Modeling Framework

Heiko Kern und Stefan Kühne
Universität Leipzig, Institut für Informatik,
Betriebliche Informationssysteme
Johannisgasse 26
04103 Leipzig
{kern, kuehne}@informatik.uni-leipzig.de

Abstract: Die Architektur integrierter Informationssysteme (ARIS) ist ein technischer Raum für den Bereich des Geschäftsprozessmanagements. Ein wesentlicher Bestandteil von ARIS ist der Modelltyp Ereignisgesteuerte Prozesskette (EPK), der in verschiedenen ARIS-Modellierungsmethoden Verwendung findet. Die Interoperabilität von ARIS-Modellen mit anderen Räumen zur Wiederverwendung bzw. Weiterverarbeitung ist durch ARIS-spezifische Export- und Import-Schnittstellen beschränkt.

In diesem Beitrag wird ein umfassender Syntaxdefinitons-sensitiver Export/Import-Mechanismus für ARIS-Modelle zum Eclipse Modeling Framework (EMF-Raum) in Form einer ARIS-EMF-Brücke vorgestellt. Er ermöglicht Interoperabilität mit den vielfältigen EMF-Werkzeugen zur Modellverarbeitung. Das Anwendungspotenzial der Brücke wird anhand von Transformationsszenarien für EPK-Modelle skizziert und konkret am Beispiel der Syntaxprüfung mit ATL demonstriert.

1 Einleitung

Die Architektur integrierter Informationssysteme (ARIS) [Sch91, Sch98a, Sch98b, STA05] ist eine Referenzarchitektur zur Unternehmensmodellierung. Sie ist in fünf Sichten (Organisations-, Daten-, Kontroll-, Funktions- und Leistungssicht) sowie in drei Abstraktionsebenen gegliedert. Diesen Sichten und Abstraktionsebenen sind verschiedene Modellierungssprachen zugeordnet. Eine zentrale Modellierungssprache ist die (Erweiterte) Ereignisgesteuerte Prozesskette (EPK) [KNS92]. Der Prozesssicht zugeordnet, dient sie der Beschreibung von Geschäftsprozessen, indem sie Elemente statischer Sichten mit einem Kontrollfluss in Beziehung setzt.

ARIS kann als ein „technischer Raum“ betrachtet werden – ein Begriff, der erstmals in [KBA02] Erwähnung findet. Als solcher stellt ARIS eine Menge von Ideen, Konzepten, Werkzeugen, Möglichkeiten und erforderlichen Fähigkeiten dar und ist einer Gruppe/Gemeinschaft zugeordnet, in der ein gewisser Konsens hinsichtlich des Wissensgebietes bzw. der akzeptierten Literatur herrscht und welche sich über Workshops und Konferenzen austauscht.

Als technischer Raum stellt ARIS eine Reihe von Werkzeugen [IDS07] zur Verfügung, wie den ARIS SOA Architect oder den ARIS Business Architect. Sie werden für verschie-

dene Zielstellungen, wie ISO-9000-Zertifizierung, Performance Management, oder der Geschäftsprozessorientierten Ausrichtung von IT-Strukturen, in verschiedenen Branchen, wie E-Government, E-Finance oder Automotive, eingesetzt. Die Werkzeuge repräsentieren eine ganzheitliche Modellierungs- und Modellverwaltungsumgebung. Zur automatisierten Verarbeitung von Modellen stehen imperative Skriptsprachen und Makros zur Verfügung, mit denen beispielsweise Auswertungsreporte erzeugt werden können.

Ein technischer Raum, der sich aufgrund offener Standards, frei verfügbarer Open-source-Implementierungen und zahlreicher Werkzeuge für die Verarbeitung von Modellen empfiehlt, ist durch das Eclipse Modeling Framework (EMF) gegeben. Beispielsweise existieren für die Umsetzung von Modell-zu-Modell- und Modell-zu-Text-Transformationen Werkzeuge unterschiedlicher Ansätze (Graph-basierter Ansatz, Relationaler Ansatz, Funktionaler Ansatz, Template-basierter Ansatz). Diese Ansätze bieten im Vergleich zum imperativen Ansatz hinsichtlich Modularität, Anpassbarkeit, Wiederverwendbarkeit oder Benutzbarkeit ein signifikantes Verbesserungspotenzial.

Für die Interoperabilität mit anderen technischen Räumen stellt ARIS XML-basierte Schnittstellen zur Verfügung. Hierzu gehört der sprachunabhängige Ansatz der ARIS Markup Language (AML), der jegliche Art von ARIS-Modellen unabhängig der zu Grunde liegenden Modellierungsmethode exportiert. Des weiteren existieren sprachabhängige Exportmöglichkeiten, die bestimmte ARIS-Modelle, die anhand einer bestimmten Modellierungsmethode erstellt wurden, in ein bestimmtes Format (bspw. BPEL4WS) exportieren. Beide Möglichkeiten sind hinsichtlich der Weiterverarbeitungsmöglichkeiten in EMF-Werkzeugen, die sich im Allgemeinen auf Modelle und deren zu Grunde liegender Sprachdefinition gleichermaßen beziehen, beschränkt.

Ein allgemeiner Ansatz ist durch die im Folgenden skizzierte und demonstrierte ARIS-EMF-Brücke [KK07] gegeben. Sie erlaubt den Export in das EMF-spezifische Format für jegliche Art von ARIS-Modellen bei gleichzeitiger Extraktion der Sprachdefinition. Im EMF-Raum können die exportierten Modelle mit den zur Verfügung stehenden Werkzeugen bearbeitet werden und bei Bedarf anschließend wieder in ARIS-Werkzeuge importiert werden. Die zur Verfügung gestellte Interoperabilität vergrößert den Entscheidungsspielraum hinsichtlich zur Verfügung stehender Werkzeuge. Die Brücke basiert ausschließlich auf Meta-sprachlichen Konzepten und Beziehungen und folgt daher einem ähnlichen Ansatz [BDD+05] vergleichbarer Brücken zwischen EMF und den Microsoft DSL Tools [BHJ+05] oder EMF und MetaGME [BBC+05]. Sie ist damit unabhängig von einer gewählten ARIS-Modellierungsmethode. Des Weiteren sichert dieser Ansatz die Einheitlichkeit und Konsistenz der Umsetzung verschiedenartiger Sprachdefinitionsansätze.

Im Folgenden werden zunächst die Sprachdefinitionselemente des EMF- und ARIS-Raums erörtert. Aufbauend auf diesen Konzepten werden Beziehungen zwischen diesen Räumen hergestellt und der Aufbau der ARIS-zu-EMF-Brücke vorgestellt. Anschließend wird das Anwendungspotenzial skizziert und die Brücke anhand der Syntaxüberprüfung von EPKs mit ATL demonstriert.

2 Eclipse Modeling Framework

Das Eclipse Modeling Framework ist ein Open-source-Framework, das die Entwicklung von Java-Anwendungen unterstützt und als Plug-In für die Eclipse-Plattform zur Verfügung steht. Es ermöglicht die Beschreibung von Datenstrukturen in Form von Datenmodellen. Die Verwaltung von Instanzen (z. B. Serialisierung, Überwachung von Änderungen) erfolgt durch aus dem Datenmodell generierte Java-Klassen oder einer EMF-Laufzeitkomponente, welche Datenmodelle dynamisch interpretiert.

Im Sinne der modellgetriebenen Software-Entwicklung [SV06] wird ein EMF-Datenmodell als Metamodell und dessen Ausprägungen bzw. Instanzen als Modelle bezeichnet. Metamodelle werden in EMF mit Hilfe von Ecore [BSM+04], dem so genannten Metametamodell, beschrieben. Durch die Ausprägungs- bzw. Instanzbeziehungen zwischen Modell, Metamodell und Metametamodell ergibt sich eine Metamodellhierarchie, die aus den drei Ebenen M1–M3 besteht. Die Ausprägungs- bzw. Instanzbeziehung zwischen den Modellen auf den einzelnen Ebenen muss nicht einer objektorientierten Instanzbeziehung entsprechen, daher wird sie im Folgenden als Konform-zu-Beziehung [Béz05] bezeichnet.

Abbildung 1 zeigt ein Beispiel für die Metamodellhierarchie. Auf M3-Ebene ist ein Ausschnitt aus Ecore dargestellt, der die zentralen Modellelemente `EClass`, `EAttribute`, `EReference` und `EDataType` zeigt. Eine EClass (Ausprägung von `EClass`) beschreibt auf M2-Ebene eine Klasse von Modellelementen mit bestimmten gemeinsamen Eigenschaften. Relationen zwischen EClasses können mit Hilfe von EReferences (Ausprägung von `EReference`) ausgedrückt werden. Weiterhin können EClasses Attribute besitzen. Diese Attribute werden durch Ausprägungen von `EAttribute` beschrieben. Der Wertebereich eines EAttributes wird durch dessen EDataType festgelegt. Dies kann beispielsweise `EString`, `EInt` oder `EBoolean` sein.

Auf M2-Ebene des Beispiels ist ein Metamodell dargestellt, welches aus der EClass „Funktion“ und „Ereignis“ besteht. Ein Ereignis steht in Relation zu einer Funktion durch die Aktiviert-Relation und umgekehrt durch die Erzeugt-Relation. Diese beiden Relationen werden durch die EReference „Aktiviert“ und „Erzeugt“ realisiert. Auf M1-Ebene ist ein entsprechend zu dem Metamodell konformes Modell dargstellt. Es besteht aus der Funktion „F1“ und dem Ereignis „E1“ und der einer Erzeugt-Referenz „Erz1“, die „F1“ und „E1“ verbindet.

3 Architektur integrierter Informationssysteme

Analog zum EMF-Raum lässt sich der technische Raum ARIS ebenso in eine Drei-Metaebenen-Hierarchie gliedern [STA05]. Auf Basis eines ARIS-spezifischen Metametamodells (im Folgenden als A3-Modell bezeichnet) sind die zur Verfügung stehenden Modellierungskonzepte definiert. Im Gegensatz zu EMF ist dieses A3-Modell jedoch nicht explizit beschrieben.

Im Sinne eines Reverse Engineering lassen sich jedoch auf Basis verfügbarer Schnittstellen die zu Grunde liegenden Konzepte und Beziehungen identifizieren und in einem ein-

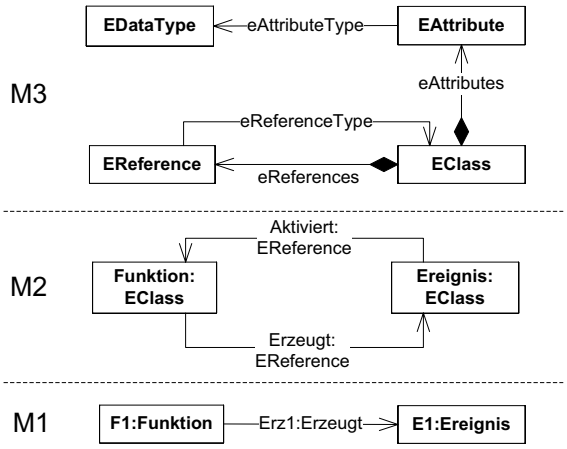


Abbildung 1: Metamodell-Hierarchie in Eclipse EMF

heitlichen Modell integrieren. Abbildung 2 zeigt die Ergebnisse einer Analyse des Werkzeugs ARIS Business Architect, insbesondere dessen Benutzungsschnittstelle, der Export-Schnittstelle AML, dem ARIS Filter-Mechanismus und die zur Verfügung stehende Programmierschnittstelle.

Im A3-Modell stellt `Modelltyp` das zentrale Element dar. Es besteht aus Kantentypen und Objekttypen, sowie deren grafischen Repräsentationen – Symbole und Linien. Modelltypen sind beispielsweise die „Ereignisgesteuerte Prozesskette“ oder das „Funktionszuordnungsdiagramm“. `Objekttyp` repräsentiert eine Menge von Objekttypen, wie „Ereignis“ oder „Funktion“, im A3-Modell. Hinterlegungsbeziehungen zwischen Modelltypen und Objekttypen legen fest, welche Modelltypen welchen Objekttypen hinterlegt werden können. Beziehungen zwischen Objekttypen werden durch Kantentypen definiert. Dabei sind Kantentypen, wie beispielsweise „aktiviert“ oder „erzeugt“, eigenständige Entitäten mit zwei mehrstelligen Referenzen auf mögliche Quell- und Zielobjekttypen. Modell-, Objekt- und Kantentypen können Eigenschaften, wie „Beschreibung“ oder „Name“ besitzen. Eigenschaften werden durch `Attributtyp` beschrieben und entsprechend von Modell-, Objekt- und Kantentypen referenziert. Der Wertebereich eines Attributtyps wird durch einen Datentyp spezifiziert. Die Festlegung der grafischen Repräsentation von Objekt-, Kanten- und Attributtypen erfolgt in Abhängigkeit von Modelltypen durch Symbole, Linien und Beschriftungen.

4 ARIS-EMF-Brücke

Ansatzpunkt für die Konstruktion einer M3-basierten Brücke ist zunächst die Konzeptabbildung der beiden Metametamodelle aufeinander (siehe Abbildung 3). Für M3-Konzepte, die sich nicht direkt aufeinander abgebilden lassen, ist die Nachbildung auf Basis zur Verfügung stehender Konzepte erforderlich. Auf Basis der M3-Ebenen-Beziehungen kön-

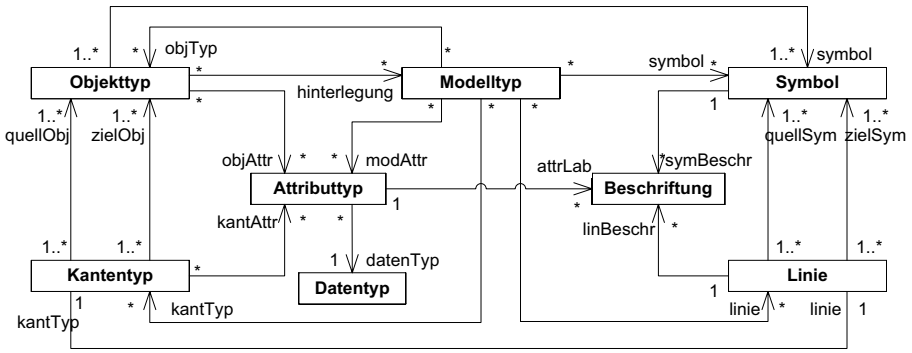


Abbildung 2: Definition der ARIS-Method auf Basis des ARIS-spezifischen Metametamodells

nen die Metamodelle auf M2-Ebene durch eine entsprechende Transformation aufeinander abgebildet werden. Der nächste Schritt besteht in der Abbildung auf M1-Ebene. Hierfür liegt ebenfalls die Konzeptabbildung auf M3-Ebene zu Grunde. Zusätzlich ist für die Definition einer entsprechenden Transformation auf M1-Ebene die Realisierung der Konform-zu-Beziehung in beiden Räumen relevant, d. h. in welcher Weise Metamodellelemente auf M1-Ebene ausgeprägt werden.

Die ARIS-EMF-Brücke bildet, wie im vorigen Absatz beschrieben, zunächst Modellelemente aus dem A3-Modell auf Ecore ab. Beispielsweise kann `Objekttyp` auf `EClass` abgebildet werden. `Kantentyp` könnte auf `EReference` abgebildet werden. Allerdings können im Gegensatz zu EReferences (ARIS-)Kantentypen Attribute besitzen. Deswegen werden Kantentypen auf EClasses abgebildet, die zwei EReferences besitzen. Diese beiden EReferences stellen die Beziehung zu den Quell- und Zielobjekten her. Weiterhin wird `Attributtyp` auf `EAttribute` abgebildet, wobei der Datentyp auf den entsprechenden EDataType abgebildet wird. Modelltypen werden ebenfalls auf EClasses abgebildet. Für die Unterscheidung der unterschiedlichen EClasses (Objekt-, Modell- und Kantentypen) werden entsprechende Oberklassen in den EMF-Metamodellen eingeführt. Aufbauend auf der Konzeptabbildung wurde eine Transformation in ARIS-Skript implementiert, welche ausgehend von einem Methodenfilter das entsprechende EMF-Metamodell in Form von XMI erzeugt. Die entgegen gesetzte Richtung – von einem EMF-Metamodell zu einem Methodenfilter – ist nur unvollständig realisierbar, da der Methodenfilter bzw. die zu Grunde liegende Methode lediglich eingeschränkt vom Nutzer geändert werden können.

Auf Basis der Abbildungen auf M3- und M2-Ebene wird eine Transformation auf M1-Ebene definiert. Hierbei ist die Speicherstruktur von ARIS zu berücksichtigen. Auf M1-Ebene sind Objekttypen als Objektdefinitionen, Kantentypen als Kantendefinitionen, Symbole als Objektausprägungen und Linien als Kantenausprägungen ausgeprägt. Für die Transformation auf M1-Ebene resultiert hieraus, dass beispielsweise Objektdefinitionen auf EClasses abzubilden sind. Die EClass auf M1-Ebene ist eine Ausprägung der EClass (M2-Ebene), die zu dem entsprechenden Objekttyp der Objektdefinition korrespondiert. Die Transformation von ARIS-Modellen zu EMF-Modellen wurde ebenfalls mit einem ARIS-Skript realisiert. Dieses Skript erzeugt in Abhängigkeit eines Methodenfilters EMF-

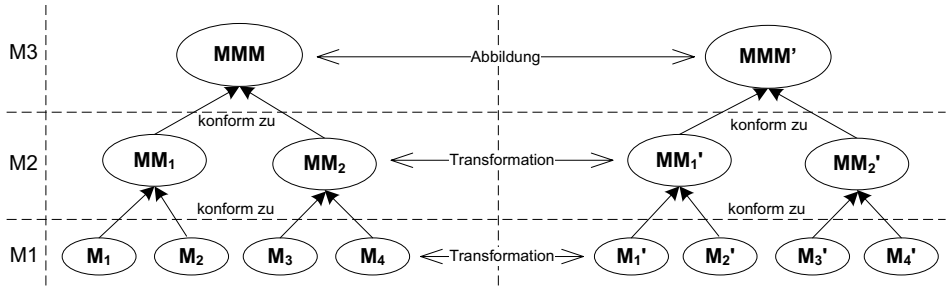


Abbildung 3: Ansatz für eine M3-Ebene-Brücke

Modelle im XMI-Format, welche konform zum Methodenfilter korrespondierenden EMF-Metamodell sind.

Im Gegensatz zur Transformation auf M2-Ebene können ARIS-Modelle nicht nur nach EMF exportiert, sondern auch wieder in ARIS importiert werden. Zum Erzeugen von ARIS-Modellen aus einem EMF-Modell wird mit Hilfe von XPand von openArchitectureWare (oAW) [Völ07] ein ARIS-Skript generiert, welches Kommandos für das Erstellen der ARIS-Modelle enthält. Die ID-Vergabe von Modellelementen erfolgt somit innerhalb des ARIS-Raums. Die eben beschriebene Brücke ist in Abbildung 4 in der Übersicht dargestellt. Eine ausführlichere Beschreibung der Transformationen ist in [KK07] zu finden.

5 Anwendungsmöglichkeiten

Die Verarbeitungsmöglichkeiten von ARIS-Modellen auf Basis von Modell-zu-Modell-Transformationen (M-zu-M-Transformation) sind vielfältig. Der EMF-Raum bietet sich für diese Aufgabe an, da eine Reihe von Transformationswerkzeugen für EMF existiert. Beispiele für solche Werkzeuge bzw. Transformationssprachen sind die ATLAS Transformation Language (ATL) [JK05], das Model Transformation Framework (MTF) [IBM04], XTend [Eff06] von oAW oder das Tiger EMF Transformation Project [BEK+06]. Diese Transformationswerkzeuge basieren auf unterschiedlichen Transformationsansätzen, wie dem Graph-basierten, relationalen, operationalen oder Template-basierten Ansatz.

Praktische Anwendungsfälle für M-zu-M-Transformation von ARIS-EPK-Modellen lassen sich hinsichtlich des Abstraktionslevels des Quell- und Ziel-Modells klassifizieren. Beispiele sind:

- die Transformation in eine Modellierungssprache auf niedrigerem Abstraktionsniveau (technische Verfeinerung, Synthese): z. B. Transformation von EPK-Modellen nach BPEL4WS,
- die Transformation in eine Modellierungssprache auf höherem Abstraktionsniveau (Reverse Engineering): z. B. Transformation von EPK-Modellen nach Wertschöpfungsdiagrammen,

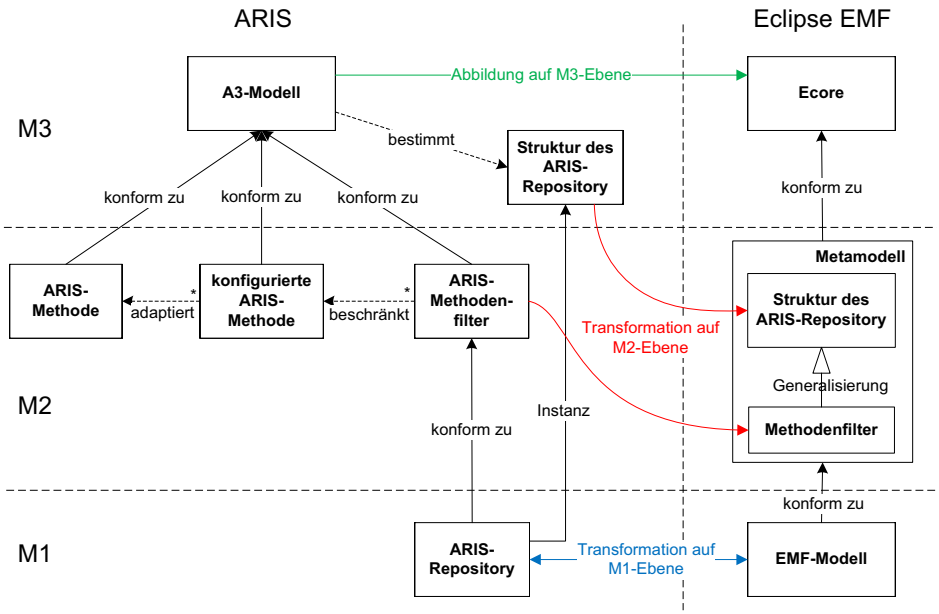


Abbildung 4: Überblick über die ARIS-EMF-Brücke

- die Transformation in eine andere Modellierungssprache auf gleichem Abstraktionsniveau (Migration): z. B. Transformation von EPK-Modellen nach UML-Aktivitätsdiagrammen,
- die Restrukturierung: z. B. Transformation von EPK-Modellen in normalisierte EPK-Modelle,
- die Extraktion bestimmter Modell-Eigenschaften (Analyse), wie die Berechnung von Prozessmetriken für EPK-Modelle oder die Überprüfung der syntaktischen Korrektheit.

Die syntaktische Überprüfung von EPKs mit Hilfe einer M-zu-M-Transformation wird im folgenden Abschnitt anhand der Transformationssprache ATL demonstriert.

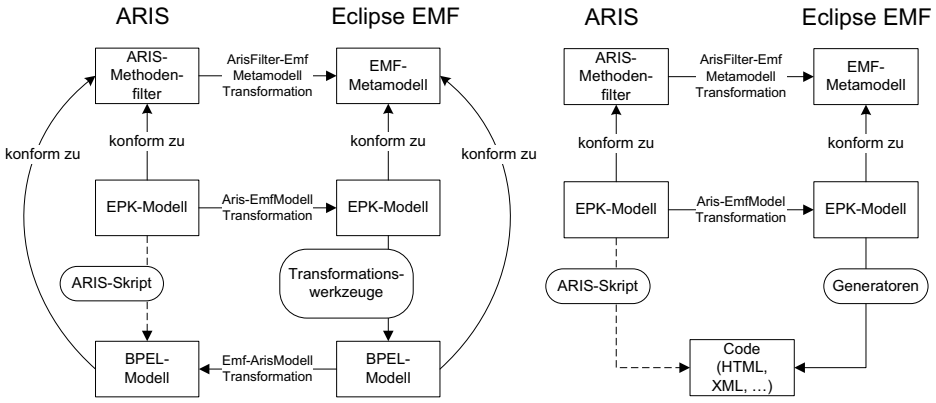
Eine weitere Möglichkeit für die Verarbeitung von ARIS-Modellen ist die Modell-zu-Text-Transformation (M-zu-T-Transformation). Im Gegensatz zur M-zu-M-Transformation, bei der das Ausgabeartefakt konform einem Ziel-Metamodell ist, liegt bei der M-zu-T-Transformation keine explizite Syntaxdefinition der Zielsprache vor bzw. wird bei der Transformation nicht berücksichtigt. Die Ausgabeartefakte sind textuell, beispielsweise Java-Code oder HTML-Code. Beispiele für Werkzeuge, die eine M-zu-T-Transformation in EMF unterstützen, sind Java Emitter Templates (JET) oder XPand [Eff06] von oAW. Ein möglicher Anwendungsfall für ARIS-EPK-Modelle ist deren Repräsentation in HTML-Form.

Neben der Transformation von ARIS-Modellen im EMF-Raum sind noch weitere Modelloperationen [Béz05] möglich, wie der Vergleich von Modellen, die Verwebung von ARIS-Modellen oder die Differenzenbildung von ARIS-Modellen. Für die Modellverwebung kann beispielsweise der ATLAS Model Weaver (AMW) [FBJ+05] eingesetzt werden und für den Modellvergleich EMF Compare [BMT07] aus dem Eclipse Modeling Framework Technology Project.

Die genannten Verarbeitungsmöglichkeiten von ARIS-Modellen im EMF-Raum erfordern den Einsatz der ARIS-EMF-Brücke. Dabei kann zwischen zwei unterschiedlichen Szenarien unterschieden werden. Zum einem kann die Brücke eingesetzt werden, um ARIS-Modelle in den EMF-Raum zu exportieren, dort zu verarbeiten und anschließend wieder in den ARIS-Raum zu importieren.

Ein solches Szenario ist in Abbildung 5(a) beispielhaft dargestellt. Hierbei wird ein Methodenfilter für EPK- und BPEL-Modelle zu Grunde gelegt. Die technische Verfeinerung von EPKs zu korrespondierenden BPEL-Modellen kann im ARIS-Raum durch ARIS-Skript realisiert werden. Die Nutzung der Brücke ermöglicht alternativ den Einsatz spezieller Transformationswerkzeuge. Erfahrungen zeigen, dass für diese Aufgabe auf Modelltransformation spezialisierte Werkzeuge im Vergleich zu transformationsunspezifischen imperativen Ansätzen Verbesserungspotenziale hinsichtlich Kompaktheit, Modularisierbarkeit, Anpassbarkeit und Komponierbarkeit eröffnen.

Im zweiten Anwendungsszenario der Brücke (siehe Abbildung 5(b)) werden ARIS-Modelle aus ARIS exportiert. Auf einen Re-Import der Modelle wird verzichtet. Die Modelle werden zur Verarbeitung in den EMF-Raum exportiert, in dem die Export-Funktion alternativ zu ARIS-Skript implementiert wird. Dabei können sowohl M-zu-M-Transformations- oder M-zu-T-Transformationswerkzeuge (Generatoren) zum Einsatz kommen.



(a) ARIS zu EMF mit Re-Import

(b) ARIS zu EMF

Abbildung 5: Anwendungsszenarien der ARIS-EMF-Brücke

Tabelle 1: EPK-Methodenfilter – Objekttypen, Symbole und Linien des EPK-Modelltyps

(a) Objekttypen und Symbole

Objekttyp	Symbol
Ereignis	Ereignis
Funktion	Funktion

(b) Linien und deren Symbole

Quellsymbol	Linie	Zielsymbol
Ereignis	aktiviert	Funktion
Funktion	erzeugt	Ereignis

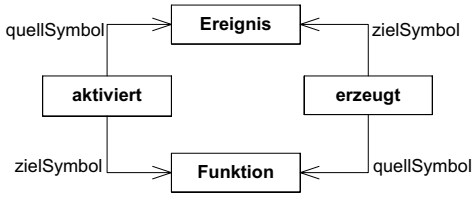


Abbildung 6: Ausschnitt aus dem EMF-Metamodell des EPK-Methodenfilters

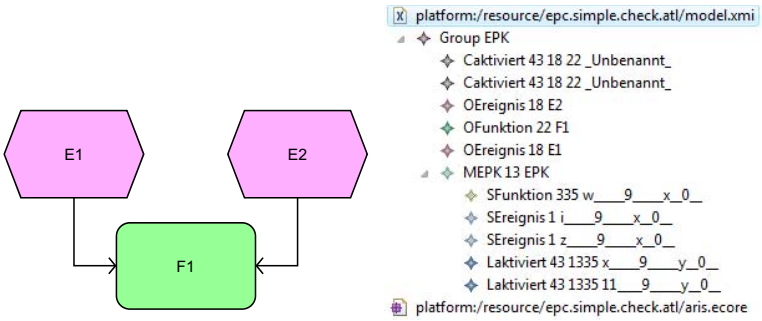
6 Syntaxüberprüfung von EPKs mit ATL

In diesem Kapitel wird auf die Analyse, speziell die Syntaxüberprüfung, von ARIS-EPK-Modellen mit ATL eingegangen. Hierzu wird zunächst die Anwendung der Brücke und anschließend die Syntaxüberprüfung demonstriert.

Für die Anwendung der ARIS-EMF-Brücke ist zunächst ein entsprechender Methodenfilter auszuwählen bzw. zu definieren, der es ermöglicht, EPK-Modelle zu erstellen. Dieser Methodenfilter besteht aus den in Tabelle 1 aufgelisteten Objekttypen, Symbolen und Linien. Zur Vereinfachung und besseren Beschreibung wurde auf die Symbole UND-Regel, ODER-Regel und XOR-Regel verzichtet. Dieser Methodenfilter wird mit dem ARIS-Skript „ArisFilter-EmfMetamodell-Transformation“ zu einem EMF-Metamodell transformiert. Ein Ausschnitt aus dem erzeugten EMF-Metamodell, welcher die Beziehungen zwischen den Symbolen darstellt, ist in Abbildung 6 abgebildet. Als nächstes können ARIS-EPK-Modelle mit der „Aris-EmfModell-Transformation“ zu einem entsprechenden EMF-Modell transformiert werden. Zur Demonstration der Transformation ist in Abbildung 7(a) eine syntaktisch nicht korrekte EPK in ARIS und die dazugehörige EPK in EMF in Abbildung 7(b) dargstellt. Ausgehend von den Regeln für korrekte EPKs, die in [NR02] beschrieben sind, werden die folgenden zwei Regeln verletzt:

1. Funktionen besitzen genau eine eingehende und genau eine ausgehende Kante.
2. Es gibt mindestens ein Start- und mindestens ein Endereignis.

Die Implementierung der beiden Regeln in der ATLAS Transformation Language wird im Folgenden beschrieben. Eine Transformationsdefinition in ATL wird als Modul bezeichnet. Dieses enthält einen Header, Import-Anweisungen, Helper und Regeln. Der Header deklariert das Quell- und Zielmodell, wobei OUT das Zielmodell und IN das Quellmodell bezeichnet. Das Quellmodell und das dazugehörige Metamodell sind das nach EMF



(a) Eine EPK als ARIS-Modell (b) Eine EPK als EMF-Modell

Abbildung 7: Eine syntaktisch nicht korrekte EPK in ARIS und EMF

überführte ARIS-EPK-Modell und der EPK-Methodenfilter. Das Zielmodell wird während der Transformation erstellt. Das dazugehörige Metamodell „Problem“ besteht im Wesentlichen aus der EClass „Problem“ und den drei Attributen „description“, „severity“ und „location“. Mit Import-Anweisungen können beispielsweise andere Module importiert werden. Dies wird allerdings nicht benötigt. Helper sind Hilfsoperationen, die in einer Regel verwendet werden können. In Abbildung 8 Zeile 3 ist ein Helper beschrieben, der aus einem ARIS-Modell alle Symbole selektiert und als Menge zurückliefert. Für die vollständige Implementierung werden weitere Helper benötigt, die alle Linien, Zielsymbole, Quellsymbole, Ereignis-Symbole, Startereignis-Symbole, Endereignis-Symbole, ausgehende Linien von Symbolen und eingehende Linien von Symbolen selektieren. Diese sind analog dem ersten Helper und werden deswegen nicht näher beschrieben.

In ATL gibt es zwei Arten von Transformationsregeln, zum einen die so genannte Called-Rule und zum anderen die Matched-Rule. Called-Rules können manuell aufgerufen werden. Sie werden in diesem Beispiel nicht benötigt und deswegen nicht weiter beschrieben. Hingegen wird bei einer Matched-Rule eine Bedingung spezifiziert, die festlegt, wann eine Regel ausgeführt wird. Die Regelausführungsstrategie wird dabei von ATL selbst übernommen. Die Bedingung einer Regel wird im ”from“-Teil spezifiziert, wobei zunächst festgelegt wird, auf welches Metamodellelement die Regel zutrifft. Anschließend kann optional durch einen OCL-ähnlichen Ausdruck diese Bedingung verfeinert werden. Im „to“-Teil einer Regel können dann entsprechend die Elemente im Zielmodell erstellt, wobei Attribute des Quellmodellelements an die Attribute des Zielmodellelements gebunden werden können.

In Abbildung 8 ab Zeile 8 sind die implementierten Regeln dargestellt. Die erste Regel „NoEndEvent“ wird angewendet, wenn die Menge der Endsymbole in einem EPK-Modell leer ist. Es wird ein Problemelement mit einer Beschreibung, dem Schwierigkeitsgrad und einer Stellenbeschreibung erstellt. Die danach folgenden drei Regeln sind analog der ersten Regel. „NoStartEvent“ wird bei einer leeren Menge von Startsymbolen in einem EPK-Modell angewendet. Die Dritte wird angewendet, wenn ein Funktionssymbol mehrere Eingangskanten besitzt und die Vierte, wenn ein Funktionssymbol mehrere Ausgangskanten hat. Bei der Ausführung der Transformationsdefinition finden die Regeln entsprechend

```
1  create OUT : Problem  from IN : Aris;

3  helper context Aris!Model def : getAllSymbols() :
     Set(Aris!Symbol) = self.containSymbols -> asSet();
5
   ...
7
   nodefault rule NoEndEvent {
9    from i : Aris!M_EPK_13(i.getAllEndEventSymbols()->isEmpty())
     to o : Problem!Problem(
11       description <- 'No end event exists.',
         severity <- #error,
13       location <- 'model name: ' + i.name + ' --- model id: ' + i.id)
   }
15
   nodefault rule NoStartEvent {
17   from i : Aris!M_EPK_13(i.getAllStartEventSymbols()->isEmpty())
     to -- ähnlich wie bei NoEndEvent
19 }

21 nodefault rule FunctionInput {
     from i : Aris!S_Funktion_335(i.getInputEdgesOfSymbol()->size() <> 1)
23   to -- ähnlich wie bei NoEndEvent
   }
25
   nodefault rule FunctionOutput {
27   from i : Aris!S_Funktion_335(i.getOutputEdgesOfSymbol()->size() <> 1)
     to -- ähnlich wie bei NoEndEvent
29 }
```

Abbildung 8: Implementierung in ATL

Anwendung und es entsteht ein Problemmodell mit den zwei Problemen, dass es kein Endereignis und eine Funktion mit mehr als einer Eingangskante gibt.

7 Zusammenfassung

In diesem Beitrag wurde eine Brücke zwischen den technischen Räumen ARIS und EMF vorgestellt. Hierzu wurden die Sprachdefinitionsmöglichkeiten beider Räume untersucht und zueinander in Beziehung gesetzt. Darauf aufbauend konnten Transformationen auf Metamodell- und Modellebene definiert werden. Die resultierende Brücke erhöht die Interoperabilität zwischen beiden Räumen und ermöglicht die Verarbeitung von ARIS-Modellen im EMF-Raum.

Am Beispiel der EPK wurde das Anwendungspotenzial von EMF-Werkzeugen skizziert und am Beispiel der Syntaxüberprüfung demonstriert. Der Raum-Wechsel von ARIS nach EMF lohnt sich genau dann, wenn auf Basis zur Verfügung stehender Werkzeuge Modellverarbeitungsaufgaben im EMF-Raum effizienter umgesetzt werden können. Die unter-

schiedlichen Transformationsmöglichkeiten bieten Verbesserungspotenziale hinsichtlich Kompaktheit, Modularisierbarkeit, Anpassbarkeit und Komponierbarkeit der Implementierungen.

Die in diesem Beitrag vorgestellte Brücke demonstriert die prinzipielle Funktionsfähigkeit des gewählten Ansatzes. Die Brücke lässt sich in verschiedener Hinsicht verbessern und erweitern. Beispielsweise wurde die Brücke hinsichtlich der Generierung graphischer Modellierungswerkzeuge (auf Basis des Eclipse Graphical Modeling Frameworks) erweitert, was die graphische Visualisierung von ARIS-Modellen im EMF-Raum ermöglicht.

Die Autoren sehen Anwendungsmöglichkeiten der ARIS-EMF-Brücke sowohl in wissenschaftlicher als auch in praktischer Hinsicht. Für konkrete Aussagen, welche EMF-Werkzeuge sich für welche EPK-Verarbeitungsaufgaben eignen, bedarf es jedoch eingehenderer Untersuchungen.

Literatur

[BBC+05] Jean Bézivin, Christian Brunette, Régis Chevrel, Frédéric Jouault und Ivan Kurtev. Bridging the Generic Modeling Environment (GME) and the Eclipse Modeling Framework (EMF). In *Proceedings of the Best Practices for Model Driven Software Development at OOPSLA'05*, San Diego, California, USA, 2005.

[BDD+05] Jean Bézivin, Vladan Devedzic, Dragan Djuric, Jean-Marie Favreau, Dragan Gasevic und Frédéric Jouault. An M3-Neutral infrastructure for bridging model engineering and ontology engineering. In *Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'2005)*, Seiten 159–171. Springer-Verlag, 2005.

[BEK+06] Enrico Biermann, Karsten Ehrig, Christian Köhler, Günter Kuhns, Gabriele Taentzer und Eduard Weiss. Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework. In Oscar Nierstrasz, Jon Whittle, David Harel und Gianna Reggio, Hrsg., *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006*, Jgg. 4199 of *LNCS*, Seiten 425–439. Springer, 2006.

[BHJ+05] Jean Bézivin, Guillaume Hillairet, Frédéric Jouault, Ivan Kurtev und William Piers. Bridging the MS/DSL Tools and the Eclipse Modeling Framework. In *Proceedings of the International Workshop on Software Factories at OOPSLA 2005*, San Diego, California, USA, 2005.

[BMT07] Cedric Brun, Jonathan Musset und Antoine Toulme. EMF Compare. `http://wiki.eclipse.org/index.php/EMF_Compare`, 2007.

[BSM+04] Frank Budinsky, David Steinberg, Ed Merks, Ray Ellersick und Timothy J. Grose. *Eclipse Modeling Framework*. the eclipse series. Addison Wesley, 2004.

[Béz05] Jean Bézivin. On the Unification Power of Models. *Software and System Modeling (SoSym)*, 4(2):171–188, 2005.

[Eff06] Sven Efftinge. OpenArchitectureWare 4.1: Extend Language Reference. Bericht, openArchitectureWare.org, `http://www.eclipse.org/gmt/oaw/doc/4.1/r25_extendReference.pdf`, 2006.

[FBJ+05] Marcos Didonet Del Fabro, Jean Bézivin, Frédéric Jouault, Erwan Breton und Guillaume Gueltas. AMW: a generic model weaver. In *Proceedings of the 1ère Journée sur l'Ingénierie Dirigée par les Modèles (IDM05)*, 2005.

[IBM04] IBM alphaWorks. Model Transformation Framework (MTF). http://www.alphaworks.ibm.com/tech/mtf, 2004.

[IDS07] IDS Scheer AG. ARIS Platform. http://www.ids-scheer.com/products, 2007.

[JK05] Frédéric Jouault und Ivan Kurtev. Transforming Models with ATL. In *Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005*, Montego Bay, Jamaica, 2005.

[KBA02] Ivan Kurtev, Jean Bézivin und Mehmet Aksit. Technological Spaces: an Initial Appraisal. In *CoopIS, DOA'2002*, 2002.

[KK07] Heiko Kern und Stefan Kühne. Model Interchange between ARIS and Eclipse EMF. In Juha-Pekka Tolvanen, Jeff Gray, Matti Rossi und Jonathan Sprinkle, Hrsg., *7th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA 2007*, 2007.

[KNS92] G. Keller, M. Nüttgens und A.-W. Scheer. Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK). Bericht Heft 89, Institut für Wirtschaftsinformatik, Universität des Saarlandes, Saarbrücken, 1992.

[NR02] Markus Nüttgens und Frank J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In J. Desel und M. Weske, Hrsg., *Promise 2002 – Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen, Proceedings des GI-Workshops und Fachgruppentreffens (Potsdam, Oktober 2002)*, Jgg. P-21 of *LNI*, Seiten 64–77, Bonn, 2002.

[Sch91] August-Wilhelm Scheer. *Architektur integrierter Informationssysteme – Grundlagen der Unternehmensmodellierung*. Springer, Berlin, Heidelberg, 1991.

[Sch98a] August-Wilhelm Scheer. *ARIS – Modellierungsmethoden, Metamodelle, Anwendungen*. Springer, 1998. (German).

[Sch98b] August-Wilhelm Scheer. *ARIS – Vom Geschäftsprozess zum Anwendungssystem*. Springer, 1998. (German).

[STA05] August-Wilhelm Scheer, Oliver Thomas und Otmar Adam. Process Modeling Using Event-Driven Process Chains. In Marlon Dumas, Wil M. P. van der Aalst und Arthur H. M. ter Hofstede, Hrsg., *Process-Aware Information Systems – Bridging People and Software through Process Technology*. Wiley, 2005.

[SV06] Thomas Stahl und Markus Völter. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[Völ07] Markus Völter. openArchitectureWare 4 – the flexible open source tool platform for model-driven software development. Bericht, openArchitectureWare, 2007.