Markus Nüttgens, Frank J. Rump (Hrsg.)

EPK 2004

Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten

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

06. Oktober 2004 in Luxemburg

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)
Hamburger Universität für Wirtschaft und Politik
Email: nuettgens@hwp-hamburg.de

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

EPK 2004 / Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. Hrsg.: Markus Nüttgens, Frank J. Rump. – Luxemburg 2004

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 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: 150 Mitglieder).

Der vorliegende Tagungsband enthält fünf vom Programmkomitee ausgewählte und auf dem Workshop präsentierte Beiträge. Jeder Beitrag wurde innerhalb der Kategorien Fachbeiträge und Diskussionsbeiträge zweifach begutachtet.

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

- EPK Semantik
- EPK Markup Language
- EPK Referenzmodelle
- EPK Website Generierung
- EPK Systementwicklungskonzepte

Der Tagungsband wurde ausschließlich in digitaler Form publiziert und ist frei verfügbar unter: http://www.epk-community.de/epk2004/epk2004-proceedings.pdf.

Wir danken den AutorInnen, den Mitgliedern des Programmkomitees und dem lokalen Organisationsteam der EMISA 2004 für die Beträge zur Realisierung des Workshops.

Hamburg und Emden, im Oktober 2004

Markus Nüttgens
Frank J. Rump

Programmkomitee

Jörg Becker, Universität Münster
Jörg Desel, Katholische Universität Eichstätt
Ekkart Kindler, Universität Paderborn
Peter Loos, Universität Mainz
Jan Mendling, Wirtschaftsuniversität Wien
Markus Nüttgens, Hamburger Universität für Wirtschaft und Politik (Vorsitz)
Andreas Oberweis, Technische Universität Karlsruhe
Frank J. Rump, Fachhochschule Oldenburg/Ostfriesland/Wilhelmshaven
Oliver Thomas, Institut für Wirtschaftsinformatik / DFKI Saarbrücken

Organisation

Markus Nüttgens, Hamburger Universität für Wirtschaft und Politik
Frank J. Rump, Fachhochschule Oldenburg/Ostfriesland/Wilhelmshaven

Inhaltsverzeichnis

On the semantics of EPCs: Efficient calculation and simulation

Nicolas Cuntz, Ekkart Kindler
Computer Science Department, University of Paderborn, Germany
cuntz|kindler@upb.de

Abstract: One of the most debatable features of *Event driven Process Chains* (EPCs) is their non-local semantics, which results in some difficulties when defining a formal semantics for EPCs. Recently, we have overcome these problems by using techniques from fixed-point theory for the definition of the semantics for an EPC, which consists of a pair of related transition relations for each EPC.

The fixed-point characterization of the semantics for EPCs, however, provides a mathematical characterization of the semantics of EPCs only. For simulating an EPC based on this semantics, we need an efficient way for calculating the corresponding pair of transition relations. A naive implementation of the underlying fixed-point approximation for calculating the transition relations, however, results in a practically useless algorithm.

In this paper, we show how to calculate the transition relations of EPCs in a more efficient way by employing different techniques and tricks from symbolic model checking for the calculation of the pair of transition relations. In addition, we analysed all kinds of simplifications of EPCs to make the calculation of the semantics more efficient, but it turned out that most of these techniques are ineffective. Still, the algorithms are fast enough for simulating practical size EPCs.

In order to demonstrate the efficiency of our algorithms and data structures, we have started an open source tool for EPCs, which we call *EPC Tools*. Right now, EPC Tools can simulate EPCs and can check some simple properties. But, EPC Tools is open for adding more sophisticated analysis and verification algorithms, and could provide a good starting point for an open source tool for the EPC community.

1 Introduction

Event driven Process Chains (*EPCs*) have been introduced in the early 90ties for modelling business processes [KNS92]. Initially, EPCs have been used informally only, without a fixed formal semantics. For easing the modelling of business processes with EPCs, the informal semantics proposed for the OR-join and the XOR-join connectors of EPCs is *non-local*. This non-local semantics, however, results in severe problems when it comes to a formalization of the semantics of EPCs and, recurrently, resulted in a debate on the semantics of EPCs [LSW98, Ri00]. It turned out that these problems are inherent to the informal non-local semantics of EPCs. In [vdADK02], we pin-pointed these arguments, which render a formal semantics that exactly captures the non-local semantics of an EPC in terms of a single transition relation impossible. But, we have shown that we can define a semantics for an EPC that consists of a pair of two correlated transition relations [Ki04b] by using fixed-point theory.

Due to their non-local semantics EPCs cannot be simply simulated by looking at the current state; rather it requires calculating the transition relations beforehand. In principle, the two transition relations defined as the semantics of an EPC can be calculated by fixed-point iteration. The problem, however, is that the calculation of the transition relations by naive fixed-point iteration is very inefficient and intractable in practice. In this paper, we will show that some techniques from *symbolic model checking* [BCM$^+$92, Mc93, CGP99] and *ordered binary decision diagrams* (ROBDDs) [Br86] in particular can be used for calculating the semantics of an EPC in a more efficient way.

We have implemented an EPC tool based on these techniques, which simulates practically relevant EPCs with a reasonable response time. Since this tool needs to calculate the transition relations of an EPC anyway, it was easy to also implement some simple semantical checks, and it should be easy to add all kinds of more sophisticated analysis and verification methods due to the fact that we use model checking techniques already for the computation of the transition relation. The tool is open source and is based on the Eclipse platform [Ecl]. Therefore, it could serve as the starting point of an open source project for a collection of EPC tools, which is the reason for calling it *EPC Tools*.

2 Syntax and Semantics of EPCs

In this section, we introduce the syntax and the semantics of EPCs as defined and motivated in [Ki04b]. The syntax, basically follows the presentation of [NR02] and the semantics is inspired by the ideas of [KNS92, NR02], but resolving the technical problems of the non-locality of EPCs as pointed out in [vdADK02].

2.1 Syntax

Figure 1 shows an example of an EPC. It consists of three kinds of nodes: *events*, which are graphically represented as hexagons, *functions*, which are represented as rounded boxes, and *connectors*, which are represented as circles. The dashed arcs between the different nodes represent the *control flow*. The two black circles do not belong to the EPC itself; they represent a *state* of an EPC. A state, basically, assigns a number of *process folders* to each arc of the EPC. Each black circle represents a process folder at the corresponding arc. In order to express some of the syntactical restrictions, we introduce a simple notation for the ingoing and outgoing arcs of a node first:

Notation 1 (Ingoing and outgoing arcs) Let N be a set of *nodes* and let $A \subseteq N \times N$ be a binary relation over N, the *arcs*. For each node $n \in N$, we define the set of its *ingoing arcs* $n_{in} = \{(x, n) \mid (x, n) \in A\}$, and we define the set of its *outgoing arcs* $n_{out} = \{(n, y) \mid (n, y) \in A\}$.

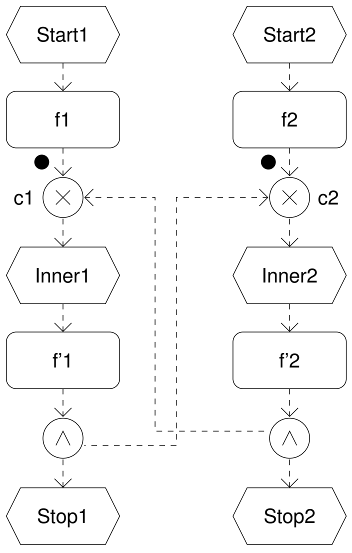


Figure 1: An EPC

A connector can be either an AND-, an OR-, or an XOR-connector, which is indicated by labelling the connector correspondingly. Each function has exactly one ingoing and one outgoing arc, whereas each event has at most one ingoing and at most one outgoing arc. A connector has multiple ingoing arcs and one outgoing arc, or it has one ingoing arc and multiple outgoing arcs:

Definition 2 (EPC) An *EPC* $M = (E, F, C, l, A)$ consists of three pairwise disjoint sets E, F, and C, a mapping $l : C \to \{and, or, xor\}$ and a binary relation $A \subseteq (E \cup F \cup C) \times (E \cup F \cup C)$ such that

- $|e_{in}| \leq 1$ and $|e_{out}| \leq 1$ for each $e \in E$,
- $|f_{in}| = |f_{out}| = 1$ for each $f \in F$, and
- either $|c_{in}| > 1$ and $|c_{out}| = 1$ or $|c_{in}| = 1$ and $|c_{out}| > 1$ for each $c \in C$.

An element of E is called an *event*, an element of F is called a *function*, an element of C is called a *connector*, and an element of A is called a *control flow* arc.

Note, that we do not consider subprocesses (called process signs in EPCs) here and that we have omitted some syntactical restrictions for EPCs. Subprocesses and the syntactical restrictions are important from a practical point of view, but they are not relevant for defining the semantics of EPCs. So, we do not formalize these restrictions here. For a complete exposition of the syntax of EPCs, we refer to [NR02] .

In the definition of the semantics, we will need to distinguish among different types of connectors: AND-, OR-, and XOR-, each of which can be either a *split* or a *join connector*.

Notation 3 (Nodes and connectors) For the rest of this paper, we fix the EPC $M = (E, F, C, l, A)$. We denote the set of all its nodes by $N = E \cup F \cup C$ and we define the following sets of connectors:

	split connectors	join connectors
$\wedge$	$C_{as} = \{c \in C \mid l(c) = \mathit{and} \wedge \lvert c_{in} \rvert = 1\}$	$C_{aj} = \{c \in C \mid l(c) = \mathit{and} \wedge \lvert c_{out} \rvert = 1\}$
$\vee$	$C_{os} = \{c \in C \mid l(c) = \mathit{or} \wedge \lvert c_{in} \rvert = 1\}$	$C_{oj} = \{c \in C \mid l(c) = \mathit{or} \wedge \lvert c_{out} \rvert = 1\}$
$\times$	$C_{xs} = \{c \in C \mid l(c) = \mathit{xor} \wedge \lvert c_{in} \rvert = 1\}$	$C_{xj} = \{c \in C \mid l(c) = \mathit{xor} \wedge \lvert c_{out} \rvert = 1\}$

2.2 States

For defining the semantics of EPCs, we need to define the *states* of an EPC first. In general, a state is an assignment of a number of process folders to the arcs of the EPC. In order to keep the state space and the transition relations finite, we consider the case of at most one folder at each arc here.

Definition 4 (State of an EPC) For an EPC $M = (E, F, C, l, A)$, we call a mapping $\sigma : A \to \{0, 1\}$ a *state* of M. The set of all states of M is denoted by Σ.

2.3 Transition relation

The semantics of an EPC defines how process folders are propagated through an EPC. Clearly, this depends on the involved node. For events and functions, a process folder is simply propagated from the incoming arc to the outgoing arc. The *transition relation* for events and functions is graphically represented in the top row of Fig. 2 (a. and b.). For connectors, the propagation of folders depends on the type of the connector (*AND*, *OR*, resp. *XOR*) and whether it is a *join* or a *split connector*. Figure 2 shows the transition relation for the connectors. For example, the AND-split connector (c.) propagates a folder from its incoming arc to all outgoing arcs. The AND-join connector (d.) needs one folder on each incoming arc, which are propagated to a single folder on the outgoing arc.

The more interesting connectors are the OR-join and the XOR-join connectors. Here, we focus on the XOR-join. An XOR-join (h.) waits for a folder on one incoming arc, which is then propagated to the outgoing arc. But, there is one additional condition: The XOR-join must not propagate the folder, if there is or there could arrive a folder on the other incoming arc. In Fig. 2.h, this is represented by a label $\nrightarrow\bullet$ at the other arc. Note that this condition cannot be checked locally: whether a folder could arrive on the other arc or not depends on the overall behaviour of the EPC. Therefore, we call the semantics of the XOR-join connector *non-local*. Likewise, the OR-join (f.) has a non-local semantics.

Note that, in this informal definition of the *transition relation*, we refer to the transition relation itself when we require that no folders should arrive at some arcs according to the transition relation. Therefore, we cannot immediately translate it to a mathematically sound definition. For now, we resolve this problem by assuming that one transition relation P is given, and we define a function $R(P)$, which defines the transition relation by referring to P instead of referring to $R(P)$.

For defining $R(P)$, we introduce some notation for restricting transition relations and for its induced reachability relation:

Notation 5 (Restriction and reachability) For some transition relation $R \subseteq \Sigma \times N \times \Sigma$, and some subset $N' \subseteq N$, we define the *restriction of* R to N' as $R|_{N'} = \{(\sigma, n, \sigma') \in R \mid n \in N'\}$. By slight abuse of notation, we define the *reachability relation* R^* of R as the reflexive and transitive closure of the binary relation $\rightarrow = \{(\sigma, \sigma') \in \Sigma \times \Sigma \mid \exists n \in N : (\sigma, n, \sigma') \in R\}$.

Next we can define $R(P)$, where we define a separate transition relation $R_n(P)$ for each node $n \in N$ of the EPC first. Then, the overall transition relation $R(P)$ is just the union of the transition relations $R_n(P)$ of all nodes n.

Definition 6 (Transition relation $R(P)$) Let P be a transition relation for an EPC M. For each node $n \in N$, we define the transition relation $R_n(P) \subseteq \Sigma \times N \times \Sigma$ as follows:

a. For $n = e \in E$ with $e_{in} = \{i\}$ and $e_{out} = \{o\}$, we define $R_e(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, e, \sigma') \in R_e(P)$ iff $\sigma(i) = 1$, $\sigma(o) = 0$, $\sigma'(i) = 0$, $\sigma'(o) = 1$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus \{i, o\}$.

a.' For $n = e \in E$ with $e_{in} = \emptyset$ or $e_{out} = \emptyset$, we define $R_e(P) = \emptyset$.

b. For $n = f \in F$ with $f_{in} = \{i\}$ and $f_{out} = \{o\}$, we define $R_f(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, f, \sigma') \in R_f(P)$ iff $\sigma(i) = 1$, $\sigma(o) = 0$, $\sigma'(i) = 0$, $\sigma'(o) = 1$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus \{i, o\}$.

c. For $n = c \in C_{as}$ with $c_{in} = \{i\}$, we define $R_c(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, c, \sigma') \in R_c(P)$ iff $\sigma(i) = 1$, $\sigma(o) = 0$ for each $o \in c_{out}$, $\sigma'(i) = 0$, $\sigma'(o) = 1$ for each $o \in c_{out}$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus (\{i\} \cup c_{out})$.

d. For $n = c \in C_{aj}$ with $c_{out} = \{o\}$, we define $R_c(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, c, \sigma') \in R_c(P)$ iff $\sigma(i) = 1$ for each $i \in c_{in}$, $\sigma(o) = 0$, $\sigma'(i) = 0$ for each $i \in c_{in}$, $\sigma'(o) = 1$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus (c_{in} \cup \{o\})$.

e. For $n = c \in C_{os}$ with $c_{in} = \{i\}$, we define $R_c(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, c, \sigma') \in R_c(P)$ iff, for some $S \subseteq c_{out}$ with $|S| \geq 1$, we have $\sigma(i) = 1$, $\sigma(o) = 0$ for each $o \in S$, $\sigma'(i) = 0$, $\sigma'(o) = 1$ for each $o \in S$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus (\{i\} \cup S)$.

f. For $n = c \in C_{oj}$ with $c_{out} = \{o\}$, we define $R_c(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, c, \sigma') \in R_c(P)$ iff, for some $S \subseteq c_{in}$ with $|S| \geq 1$, we have $\sigma(i) = 1$ for each $i \in S$, $\widehat{\sigma}(a) = 0$ for each $\widehat{\sigma}$ with $\sigma(P|_{N \setminus \{c\}})^* \widehat{\sigma}$ and for each $a \in c_{in} \setminus S$, $\sigma(o) = 0$, $\sigma'(i) = 0$ for each $i \in S$, $\sigma'(o) = 1$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus (S \cup \{o\})$.

g. For $n = c \in C_{xs}$ with $c_{in} = \{i\}$, we define $R_c(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, c, \sigma') \in R_c(P)$ iff, for some $o \in c_{out}$, we have $\sigma(i) = 1$, $\sigma(o) = 0$, $\sigma'(i) = 0$, $\sigma'(o) = 1$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus \{i, o\}$.

h. For $n = c \in C_{xj}$ with $c_{out} = \{o\}$, we define $R_c(P) \subseteq \Sigma \times N \times \Sigma$ by $(\sigma, c, \sigma') \in R_c(P)$ iff, for some $i \in c_{in}$, we have $\sigma(i) = 1$, $\widehat{\sigma}(a) = 0$ for each $\widehat{\sigma}$ with $\sigma(P|_{N \setminus \{c\}})^* \widehat{\sigma}$ and for each $a \in c_{in} \setminus \{i\}$, $\sigma(o) = 0$, $\sigma'(i) = 0$, $\sigma'(o) = 1$, and $\sigma'(a) = \sigma(a)$ for each $a \in A \setminus \{i, o\}$.

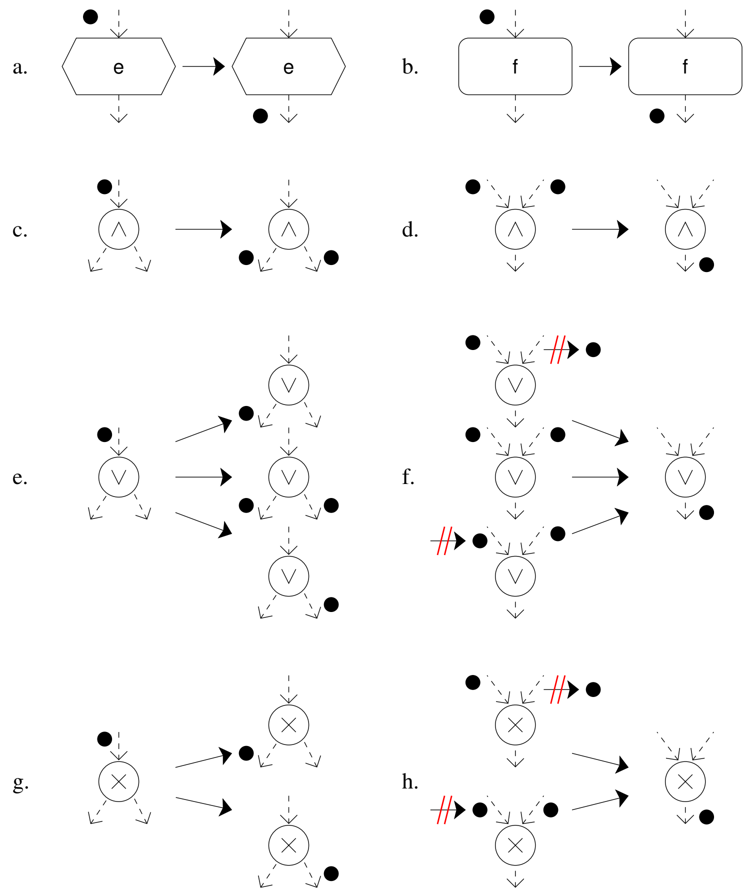


Figure 2: The transition relation $P_n = R(P)$ for the different nodes n

We define the *transition relation* $R(P) = \bigcup_{n \in N} R_n(P)$

Below, we briefly discuss the important cases f. and h. of the above definition, which concerns the non-local semantics of the OR-join and the XOR-join connectors: When there is a folder on at least one of its incoming arcs $S \subseteq c_{in}$ of an OR-join connector and no folder can arrive (according to P) on the other arcs without the occurrence of c, the folder is propagated to the outgoing arc. In order to formalize that no folder can arrive on the other incoming arcs $a \in c_{in} \setminus S$, the definition refers to the states $\widehat{\sigma}$ that can be reached from σ (with respect to P) without the occurrence of c. This is formalized by $\sigma \ (P|_{N \setminus \{c\}})^* \ \widehat{\sigma}$. The XOR-join is similar to the definition of the OR-join. Instead of selecting some set S of incoming arcs on which a folder must be present, we select exactly one incoming arc i. We require that no folder can arrive on the other incoming arcs (with respect to P) before the occurrence of c. Again, this can be formalized by using the relation $(P|_{N \setminus \{c\}})^*$.

The most important property of $R(P)$ is that it is monotonously decreasing in P, i. e. for each two transition relations P and P' with $P \subseteq P'$ we have $R(P) \supseteq R(P')$. The reason is that P occurs under a negation in the definition of $R(P)$ (see [Ki04b] for more details).

2.4 Semantics

Based on $R(P)$, we can now define the semantics of the EPC. Ideally, we would like to define it to be a fixed-point $P = R(P)$. Unfortunately, there are EPCs for which $R(P)$ does not have a fixed-point. So, we define it as a pair of transition relations P and Q such that $P = R(Q)$ and $Q = R(P)$, were P is the least such transition relation and Q is the greatest such transition relation. In [Ki04b], we have proved that this pair is uniquely defined by applying standard fixed-point theory, exploiting the fact that R is monotonously decreasing. We called P the *pessimistic transition* relation of the EPC, and we called Q the *optimistic transition* relation of the EPC. Unfortunately, P and Q can be different for some (nasty) EPCs, and we have argued that these are exactly the EPCs for which a single transition relation cannot fully capture the informal semantics of EPCs. For EPCs for which P and Q coincide the semantics exactly captures the informal semantics. Therefore, we call EPCs with $P = Q$ *clean*.

In [Ki04b], we did not bother to give an operational characterization of this semantics, since we were interested only in defining a precise semantics. But, the fixed-point theorem of Kleene immediately gives us a simple algorithm for calculating this pair (which is called fixed-point approximation):
Let $P_0 = \emptyset$ and $Q_0 = \Sigma \times N \times \Sigma$. For each $i \in \mathbb{N}$, we define $P_{i+1} = R(Q_i)$ and $Q_{i+1} = R(P_i)$. Since $R(P)$ is monotonously decreasing, we have that $P_i \subseteq P_{i+1}$ and $Q_i \supseteq Q_{i+1}$ for each i. Moreover, $\Sigma \times N \times \Sigma$ is finite, which implies that for some $i \in \mathbb{N}$ we will have $P_{i+1} = P_i$ and $Q_{i+1} = Q_i$. For this i, we have $R(P_i) = Q_{i+1} = Q_i$ and $R(Q_i) = P_{i+1} = P_i$. And this (P_i, Q_i) is the semantics of the EPC. So, starting with $P_0 = \emptyset$ and $Q_0 = \Sigma \times N \times \Sigma$ and iteratively computing the next P_{i+1} and Q_{i+1} will eventually terminate with the semantics of the EPC.

Unfortunately, an explicit representation of the transition relations P_i and Q_i and an explicit calculation of $P_{i+1} = R(Q_i)$ and $Q_{i+1} = R(P_i)$ is extremely inefficient. For realistic EPCs, there are millions of potential states Σ and billions of potential arcs in the transition relation[1]. Moreover, an explicit calculation of $R(P)$ involves a reachability analysis on P (at least for the non-local connectors). So a naive explicit implementation of the fixed-point approximation does not work in practice.

3 Calculating the transition relations

In the previous sections, we have rephrased the semantics of EPCs in an operational way. Next, we will show how the two transition relations can be calculated in a more efficient way. To this end, we will use techniques from symbolic model checking and ordered binary decision diagrams. We use formulas and temporal formulas for representing the transition relations $R_n(P)$, and we will show how these formulas can be used for efficiently calculating the semantics of the underlying EPC.

[1]Note that not all of these states will be reachable in the final semantics; but they will be necessary in the calculation of the semantics.

3.1 Representing $R_n(P)$

Let us start with the semantics of an AND-split connector, with ingoing arc i and outgoing arcs $o_1, \dots, o_n$. In order to define the corresponding behaviour, we assume that i and $o_1, \dots, o_n$ are boolean variables. The values of these variables represent the state before the transition, where value true means that there is a process folder on the corresponding arc, and value false means that there is no process folder. Moreover, we assume that, for each variable, there is a primed version i' and $o'_1, \dots, o'_n$, which represent the state after the transition. With this notation and understanding, the behaviour of the AND-split can be expressed by the following formula:

$$i \wedge \neg o_1 \wedge \dots \wedge \neg o_n \;\wedge\; \neg i' \wedge o'_1 \wedge \dots \wedge o'_n$$

This formula exactly captures the fact that there must be a folder on the ingoing arc i of the AND-split and there must be no folders on the outgoing arcs $o_1, \dots, o_n$ before firing the AND-split; and, after firing the AND-split, the ingoing arc has no folder anymore, but the outgoing arcs have a folder each. Altogether the formula is an immediate translation of Def. 6 (c), where we assume that variables not occurring in the formula do not change.

Altogether, we can apply this standard technique [CGP99, HR00] for defining the behaviour of all EPC nodes with a local semantics. The complete list of formulas for all connectors is shown below, where, for simplicity, we assume that connectors have at most two input and output arcs:

a. / b. For $n \in E \cup F$ with $n_{in} = \{i\}$ and $n_{out} = \{o\}$, the formula for $R_n(P)$ is $i \wedge \neg o \wedge \neg i' \wedge o'$.

c. For $n = c \in C_{as}$ with $c_{in} = \{i\}$ and $c_{out} = \{o_1, o_2\}$, the formula for $R_n(P)$ is $i \wedge \neg o_1 \wedge \neg o_2 \wedge \neg i' \wedge o'_1 \wedge o'_2$.

d. For $n = c \in C_{aj}$ with $c_{in} = \{i_1, i_2\}$ $c_{out} = \{o\}$, the formula for $R_n(P)$ is $i_1 \wedge i_2 \wedge \neg o \wedge \neg i'_1 \wedge \neg i'_2 \wedge o'$.

e. For $n = c \in C_{os}$ with $c_{in} = \{i\}$ and $c_{out} = \{o_1, o_2\}$, the formula for $R_n(P)$ is $i \wedge \neg(o_1 \wedge o_2) \wedge \neg i' \wedge (o_1 \Rightarrow o'_1) \wedge (o_2 \Rightarrow o'_2) \wedge (o'_1 \neq o_1 \vee o'_2 \neq o_2)$.

g. For $n = c \in C_{xs}$ with $c_{in} = \{i\}$ and $c_{out} = \{o_1, o_2\}$, the formula for $R_n(P)$ is $i \wedge \neg(o_1 \wedge o_2) \wedge \neg i' \wedge (o_1 \Rightarrow o'_1) \wedge (o_2 \Rightarrow o'_2) \wedge (o'_1 \neq o_1 \; xor \; o'_2 \neq o_2)$.

The formulas for the OR- and the XOR-split connectors are a bit more involved. For the OR-split connector (cf. e.), it is required that no outgoing arcs has less folders than before and at least one has more, which can be formulated in terms of a implication $o \Rightarrow o'$ (i. e. if there is a folder on o in the source state of the transition then there is a folder on o in the target state of the transition).

For the XOR-split (cf. g.) connector, we also require that no outgoing arc has less folders than before and exactly one arc has one more. Some formulas are a bit involved, but, in principle, there is no problem with these formulas for the local connectors, because the transition relation $R_n(P)$ does not refer to P.

But, how about the formulas for the non-local operators? For these connectors the definition of $R_n(P)$ refers to P. So, we need to refer to P in the formula for $R_n(P)$ somehow. To this end, we use a temporal logic formula that is interpreted on the transition relation P. Since we use very simple formulas only, we do not bother to introduce temporal logic in full detail. The only temporal operator needed is the CTL operator EF: For some formula φ the temporal formula $EF\varphi$ is true in exactly those states from which a state can be reached (with respect to P) in which φ is valid. This way, we can express that no folder can arrive on some arc i by the formula $\neg EFi$.

With this temporal formula, it is easy to express the behaviour of the XOR-join connector: For an XOR-join connector with the two incoming arcs i_1 and i_2 and one outgoing arc o,

$$((i_1 \wedge \neg EFi_2) \vee (\neg EFi_1 \wedge i_2)) \;\wedge\; \neg o \wedge \neg i'_1 \wedge \neg i'_2 \wedge o'$$

precisely captures its behaviour. The formulas $\neg EFi_1$ resp. $\neg EFi_2$ guarantee that a transition does occur only when no folder can arrive from the other arc, respectively.

For the OR-join connector, the transition relation is similar. It requires that there is one folder on one incoming arc and, if there is no folder on the other incoming arc no folder can arrive at this arc anymore. Altogether, we define:

f. For $n = c \in C_{oj}$ with $c_{in} = \{i_1, i_2\}$ $c_{out} = \{o\}$, the formula for $R_n(P)$ is $((i_1 \wedge i_2) \vee (i_1 \wedge \neg EFi_2) \vee (\neg EFi_1 \wedge i_2)) \wedge \neg o \wedge \neg i'_1 \wedge \neg i'_2 \wedge o'$

h. For $n = c \in C_{xj}$ with $c_{in} = \{i_1, i_2\}$ and $c_{out} = \{o\}$, the formula for $R_n(P)$ is $((i_1 \wedge \neg EFi_2) \vee (\neg EFi_1 \wedge i_2)) \wedge \neg o \wedge \neg i'_1 \wedge \neg i'_2 \wedge o'$

Experts in model checking may be a bit concerned about mixing primed variables and temporal operators in a single formula. Usually, there are transition formulas that may contain primed variables, but no temporal operators, and there are temporal formulas that must not contain primed variables. A transition formula or a set of transition formulas represents the underlying system; the temporal formulas represent properties to be verified for that system. Though uncommon, in principle, there is no harm in mixing primed variables and temporal operators in a single formula. It defines a new transition relation based on a given transition relation, which is exactly what we need for calculating $R_n(P)$.

3.2 Calculating the transition relations

Next, we will discuss how to calculate the two transition relations that actually represent the semantics of an EPC, where we assume that the EPC has the local nodes $l_1, \ldots, l_j$ and the non-local nodes $n_1, \ldots, n_k$, and $g_1, \ldots, g_j$ are the formulas representing the transition relations $R_{l_i}(P)$ for the local nodes, and $h_1, \ldots, h_k$ are the formulas representing the transition relations $R_{n_i}(P)$ for the non-local nodes.

Let us first discuss the operations from model checking that we need for this calculation. In symbolic model checking, a transition relation given as a formula (with primed variables) is transformed into a data structure that is called a *reduced ordered binary decision* *diagram*[2] (ROBDD), which have the nice feature that equivalent formulas will have exactly the same ROBDD representation. For a formula f with primed variables without temporal operators, there is a standard procedure for this transformation [CGP99, HR00]. We denote this procedure by f.toROBDD(), which is close the corresponding methods of our object oriented model checker MCiE [Ki04a].

Formulas with primed variables and temporal variables are very uncommon. So there is no standard procedure for converting it to an ROBDD. But, there is a standard procedure for calculating an ROBDD representing the set of states of a transition system in which a given temporal formula is true. We assume that the transition system is given as a set P of ROBDDs representing the transitions of the system. This procedure can be easily extended to formulas that contain primed variables. For such a formula f and an ROBDD-representation P of the transition relation, f.toROBDD(P) denotes the resulting ROBDD.

Given some transition system (represented as a set of ROBDDs) $\mathtt{P_{curr}}$, we can calculate $\mathtt{P_{next}} = R(\mathtt{P_{curr}})$ as follows:

```
$\mathtt{P_{next}}$:= { $g_1$.toROBDD(),..., $g_j$.toROBDD() };
for $i$:= 1 to k do
    $\mathtt{P_{next}}$:= $\mathtt{P_{next}}$.add($h_i$.toROBDD($\mathtt{P_{curr}}$));
```

In the first line, we insert all the transitions of the local nodes to $\mathtt{P_{next}}$; in the loop, we add the transition relation for each non-local node to $\mathtt{P_{next}}$. To be precise, the calculation is a bit more involved: In order to exactly capture the semantics formalized in Sect. 2.4, we must switch off the transition relation corresponding to node n_i for calculating the next transition relation for node n_i. Since this is a minor technical detail, we do not include this into the presented pseudo code.

Based on this code, we can easily start the calculation of the transition relations P_i and Q_i as stated in the formal definition of the semantics in Sect. 2: $P_0 = \emptyset$ and $Q_0 = \Sigma \times N \times \Sigma$ and $P_{i+1} = R(Q_i)$ and $Q_{i+1} = R(P_i)$. In order to save computation time, we do not calculate every P_i and every Q_i, rather we calculate $Q_0, P_1, Q_2, P_3, \dots$ in a zig-zag way. As stated before, we will eventually end up with $P_{i+2} = P_i$ and $Q_{i+2} = Q_i$; in order to detect this point, we need to store the last two versions of the calculated transition relations and compare them to the next one. When they are equal, we have calculated the two transition relations that represent the semantics of the EPC.

Altogether, the algorithm for calculating the semantics of an EPC looks as follows.

[2] Often reduced ordered binary decision diagrams are called ordered binary decision diagrams (OBDDs) or even binary decision diagrams (BDDs) only. We stick to the term ROBDD throughout this paper, however.

```
P_curr := { false }; // $P_0$
P_next := { true }; // $Q_0$
step:= 1;

repeat
   P_prev := P_curr;
   P_curr := P_next;
   step:= step + 1;

   // P_next = $R$(P_curr)
   P_next := { $g_1$.toROBDD(),..., $g_j$.toROBDD() };
   for $i$:= 1 to k do
      P_next := P_next.add($h_i$.toROBDD(P_curr));

until P_next == P_prev;
```

Upon termination $\mathtt{P_{curr}}$ and $\mathtt{P_{next}}$ contain the two transition relations for the EPC. The question, however, is which of them is the pessimistic and which is the optimistic transition relation. In order to decide this, we use the `step` counter. If it is odd, $\mathtt{P_{curr}}$ represents the pessimistic transition relation and $\mathtt{P_{next}}$ is the optimistic transition relation; otherwise, it is the other way round.

3.3 Simulation

Once we have calculated the two transition relations for an EPC, it is easy to simulate it. For some given state, we must calculate all nodes that can propagate a process folder (according to the pessimistic or according to the optimistic transition relation). In that case, we call the corresponding node *enabled* in this state. Since we store the calculated ROBDDs P_x for each transition relation for node x separately, checking the enabledness is simple. Let `enabled` be the CTL formula $EXtrue$, which is valid in all states for which the underlying transition relation has a successor. Then `enabled.toROBDD(`P_x`)` represents all those states in which the node is enabled.

When the user wants to fire an enabled transition, the simulator explicitly removes and adds the folders in the current state according to semantics of the corresponding node. It is not necessary to use ROBDDs here because only the enabledness of a node is non-local. The propagation of the folders itself is local.

3.4 Implementation

It is easy to implement the above algorithms based on some standard ROBDD package. The only tricky part might be the mixed occurrence of primed variables and temporal operators in formulas. Since our own *Model Checking in Education* (MCiE) project immediately supports this kind of formulas, we implemented the algorithm based on MCiE. Though MCiE is implemented in Java and efficiency is not MCiE's highest priority, the first experiments with this algorithm were surprisingly good. Without further optimizations, it worked reasonably well on small EPCs. For calculating the semantics for larger EPCs, however, we had to come up with some optimizations, which will be discussed below.

In principle, we could use any other ROBDD package for implementing the calculation of the semantics of EPCs. Since we were heading for an Eclipse based tool (see Section 5) and MCiE was available in Java, however, we still use MCiE.

4 Optimizations

As mentioned above, we had to apply several tricks and optimizations in order to compute the semantics of larger EPCs. In our discussion, we distinguish between two different kinds of optimizations.

The first kind tries to exploit properties of the semantics of EPCs in order to reduce and to simplify them. The idea is to calculate the semantics of a simpler and smaller EPC and, based on this information, simulate the original EPC. These optimizations have been investigated in [Cu04]. Unfortunately, there are many negative results, which basically can be considered as a backfiring of the non-local semantics of EPCs. The non-local semantics of EPCs seems to have many nasty side effects and renders many ideas for optimizations impossible – except for very trivial ones.

The second kind is a smart application and combination of optimization techniques generally known from model checking. It turned out that these techniques were much more effective than the ones for EPCs and could be used in combination with the ones for EPCs.

Note that, in spite of all our optimizations, the worst case complexity of our algorithms is still very bad: it is exponential. It is an interesting open question whether this is inherent to the semantics of EPCs or not. But, we feel that, again, this worst case complexity cannot be avoided because of the non-local semantics of EPCs. But, our experimental results have shown that, for many practical examples, we can calculate the semantics of many practically relevant EPCs in a reasonable time (see Sect. 4.3).

4.1 EPC techniques

We start with a brief discussion of techniques that exploit the properties of EPCs.

Eliminating chains It is clear that reducing the size of an EPC also reduces the complexity of the simulation problem. For our model checking algorithm, the number of arcs of the simulated EPC is essential, because the computation time is exponential in the number of variables respectively arcs.

One possible approach is to simplify an EPC by eliminating chains of nodes that do not influence the semantical behaviour of other nodes. Obviously, a sequence of consecutive event and function nodes such as the ones shown in Fig. 3 (labelled Event and Function) can be omitted when computing the enabledness of the XOR-join connectors. We call this optimization *chain elimination*,

We can apply chain elimination, when the following two conditions are satisfied:

1. In the considered state, there are no process folders on the arcs eliminated by this simplification.

   It is obvious that, otherwise, a process folder in the predecessor set of an XOR-join connector which would have potentially influenced the behaviour of the XOR-join in the original EPC would be missing in the simplified EPC.

   Note that this condition implies that we can omit only those arcs from a chain that do not have a folder on them. Therefore, chain elimination depends on the considered state of the EPC. For simulation, this is no serious problem because we can compute another simplified EPC each time the state has changed. Since the simplified EPC is much smaller than the original one, we can hope that the fixed-point computation is significantly faster for the reduced EPC. For the analysis and, in particular, for checking whether the semantics of an EPC is clean, however, we cannot apply this chain elimination technique directly.

2. In order to correctly apply chain elimination, it is necessary that in no reachable state of the reduced EPC, a node is blocked because of a process folder on one of its outgoing edges. We call such states *contact situations*. The problem with *contact situations* is, that the simplified EPCs tend to have more contact situations as compared to the original EPCs. In this situation, the behaviour of the original and the simplified EPC are different. The simplified EPC is blocked, whereas the original version could still fire. Therefore, we cannot use the simplified version for simulation the original one. Fortunately, it is easy to calculate whether the simplified EPC has reachable contact situations, which provides us an a posteriori condition, whether chain elimination can be applied. In that case, we can switch back to calculating the semantics of the original EPC, which of course is less efficient.

If both requirements have been checked, the simulator can use the transition relation computed for the simplified EPC to determine whether an XOR-join resp. an OR-join connector is enabled in the original EPC or not (other nodes can be checked locally anyway). Because we only eliminate event and function nodes, those connectors are contained in the reduced EPC.

The main disadvantage of the chain elimination approach is that it cannot be applied for arbitrary EPCs because of the above requirements. Also, chain elimination does not allow us to calculate the complete semantics of an EPC. Therefore, it can be used for simulation only; it cannot be used for our analysis and verification algorithms.

Syntactical restrictions An other idea for simplifying the simulation problem was to identify some restricted classes of EPCs for which no fixed-point iteration would be nec-

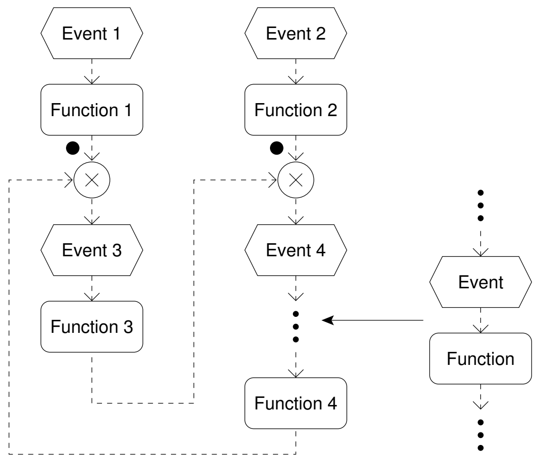


Figure 3: The example used for the measurements in Fig. 4

essary. For example, we considered EPCs without cycles on non-local nodes such as the ones shown in Fig. 1, or EPCs that are constructed from clean EPC constructs only. We hoped that we could calculate the semantics of EPCs from these sub-classes in a much more efficient way. Unfortunately, it turned out that this hope was in vain, and we found some nasty counter-examples, which spoiled this approach. A detailed discussion of these negative results can be found in [Cu04].

4.2 Model checking techniques

There are many techniques that make model checking more efficient. Using ROBDDs as a representation for sets of states and for the transition relations is one of them. It is only this choice, that made our algorithms work for small examples. In addition to using ROBDDs, we used two other techniques: optimization of the variable order and partitioning of the transition relation.

Variable order It is well-known that the size (number of nodes) of the ROBDDs representing some boolean function or formula strongly depends on the chosen variable order. In turn, the computation time of the operations on ROBDDs depends on the size of the ROBDDs. So, it is important to find a good variable order for efficiently calculating the semantics of EPCs. One heuristic for a good variable order is that related variables should be close to each other in the variable order. For EPCs, it is quite easy to identify those variables (arcs) that are related: Two variables resp. arcs are related, when they are attached to the same node. The problem, however, is that each arc belongs to two nodes; so it is impossible to have all related variables close to each other in the variable order, in particular, when the EPC has cycles in its control flow arcs. In order to calculate a good variable order, we thought of some sophisticated schemes. But, in the end, it turned out that a simple breadth first traversal of all nodes starting from the start events of the EPC

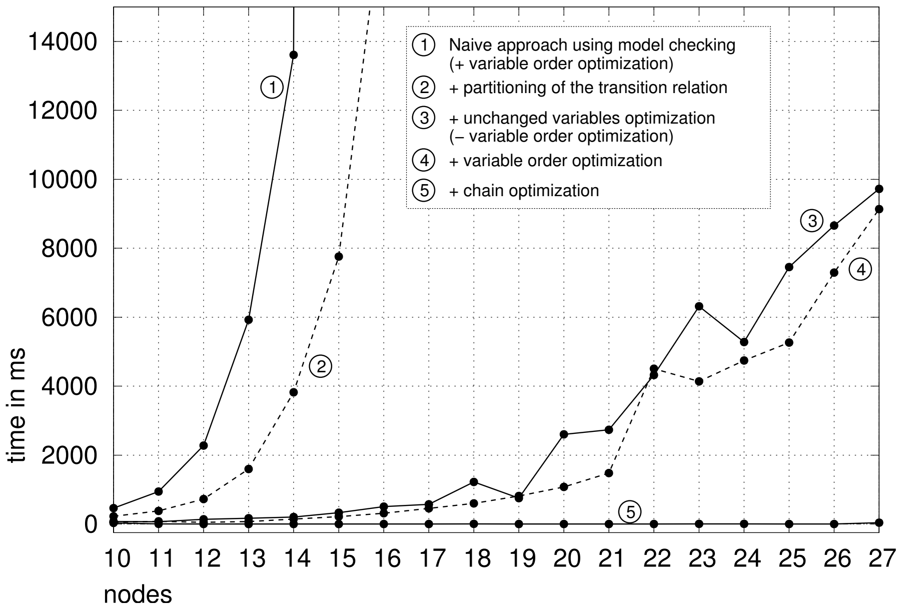


Figure 4: Benefit of the implemented optimization techniques (compare to Fig. 3)

provided a variable order with the best results.

Though this variable order provided satisfactory results, we feel that there is some room for further improvement. But, this needs further investigations.

Partitioning of the transition relation In the algorithm for calculating the transition relations of an EPC, we distinguish the ROBDDs for the transition relations for each node of the ROBDD. It is well known that this results in much less nodes for representing the transition relation than for representing all transitions within a single ROBDD.

In order to make these ROBDDs even smaller, we imposed one additional assumption on the formulas representing the transition relation: we assume that all variables not occurring in the formula do not change. Expressed in a naive way, this means adding the formula $a_1 = a_1' \wedge a_2 = a_2' \wedge \ldots \wedge a_n = a_n'$ for all variables that are not touched by this node. Adding this formula explicitly to the transition relation, however, would result in much bigger ROBDDs, which in turn would result in much longer computation times. Therefore, we did not add this formula to the representation of the transition relation, but we implemented the procedure for calculating EX within the ROBDD library in such a way that these variables were implicitly assumed to be unchanged. This *unchanged variables optimization* resulted in significantly better computation times.

4.3 Measurements

In order to illustrate the benefits of the optimizations, Fig. 4 shows the computation times for calculating the semantics of the example of Fig. 3 for the different optimization techniques. In order to see the influence of the size of the EPC, we measured the computation time for different numbers of nodes on the chain between Event 4 and Function 4. The x-axis represents the number of nodes of the resp. EPC, the y-axis shows the computation times for the different optimizations. The first graph shows the computation time without partitioning the transition relations. The second graph shows the computation time with partitioning, but without the improvement for unchanged variables. The third graph shows the time when incorporating also the optimization for unchanged variables in the transition relations. The fourth graph shows the time with an optimized variable order. Note that partitioning the transition relation along with an explicit algorithm for unchanged variables makes a significant difference in the computation times.

The fifth graph shows that chain elimination can drastically improve the simulation of EPCs. Note, however, that this example is a bit misleading because it was chosen to show the positive effect of chain elimination. In other examples, the figures are not as impressing and, in many situations, chain elimination is not applicable at all (see discussion above).

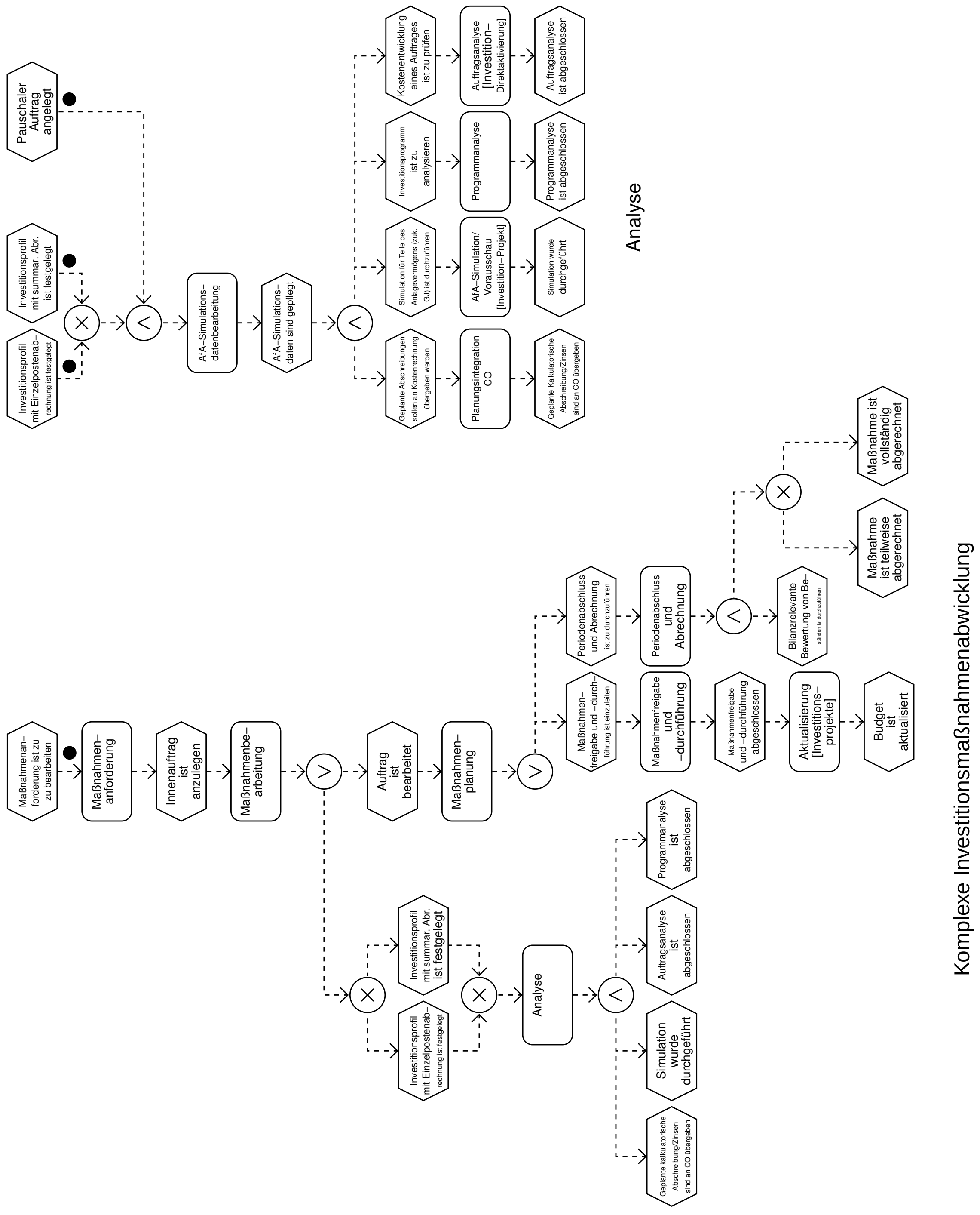
The above figures come from a technical example. In order to give an impression of computation times for real-world examples, we have considered two examples from the SAP reference processes of the ARIS Toolset[3], which are shown in Fig. 5. The calculation of the semantics of the first EPC took 13 ms; for the second, it took 4.015 s. With the transition relation computed, the other operations (simulation and analysis) can be done in virtually no time.

5 EPC Tools

The algorithm for calculating the semantics of an EPC and for simulating an EPC based on this semantics is integrated into an Eclipse [Ecl] based tool, which we call *EPC Tools*. EPC Tools can be obtained from [CuKi04] free of charge. Figure 6 shows a screen-shot of Eclipse with the EPC Tools plugin running. EPC Tools comes with a graphical editor and an interactive simulator for EPCs. Moreover, it is easy to import EPCs from other tools because EPC Tools supports the EPC exchange format *EPML* [MN04a], and there are converters between the AML format of the ARIS Toolset and EPML [MN04b].

Moreover, EPC Tools checks simple semantical properties of the EPC. For example, it indicates whether the EPC is clean, i. e. whether both transition relations coincide. This is important, because unclean EPCs can easily lead to different interpretations and should be considered harmful. EPC Tools identifies unclean EPCs right away. In addition, EPC Tools checks whether an EPC might deadlock and whether there are contact situation, i. e.

[3]ARIS Toolset is a registered trademark of IDS Scheer. For more information see `http://www.ids-scheer.com/`.



Analyse
Investitionsprofil mit Einzelpostenabrechnung ist festgelegt
Investitionsprofil mit summar. Abr. ist festgelegt
Pauschaler Auftrag angelegt
AfA–Simulationsdatenbearbeitung
AfA–Simulationsdaten sind gepflegt
Geplante Abschreibungen sollen an Kostenrechnung übergeben werden
Simulation für Teile des Anlagevermögens (zuk. GJ) ist durchzuführen
Investitionsprogramm ist zu analysieren
Kostenentwicklung eines Auftrages ist zu prüfen
Planungsintegration CO
AfA–Simulation/ Vorausschau [Investition–Projekt]
Programmanalyse
Auftragsanalyse [Investition–Direktaktivierung]
Geplante Kalkulatorische Abschreibung/Zinsen sind an CO übergeben
Simulation wurde durchgeführt
Programmanalyse ist abgeschlossen
Auftragsanalyse ist abgeschlossen
Komplexe Investitionsmaßnahmenabwicklung
Maßnahmenanforderung ist zu zu bearbeiten
Maßnahmenanforderung
Innenauftrag ist anzulegen
Maßnahmenbearbeitung
Auftrag ist bearbeitet
Maßnahmenplanung
Maßnahmenfreigabe und –durchführung ist einzuleiten
Periodenabschluss und Abrechnung ist zu durchzuführen
Maßnahmenfreigabe und –durchführung
Periodenabschluss und Abrechnung
Maßnahmenfreigabe und –durchführung abgeschlossen
Bilanzrelevante Bewertung von Beständen ist durchzuführen
Aktualisierung [Investitions–projekte]
Budget ist aktualisiert
Maßnahme ist teilweise abgerechnet
Maßnahme ist vollständig abgerechnet
Investitionsprofil mit Einzelpostenabrechnung ist festgelegt
Investitionsprofil mit summar. Abr. ist festgelegt
Analyse
Geplante kalkulatorische Abschreibung/Zinsen sind an CO übergeben
Simulation wurde durchgeführt
Auftragsanalyse ist abgeschlossen
Programmanalyse ist abgeschlossen

Figure 5: Two examples from the SAP reference processes of the ARIS Toolset

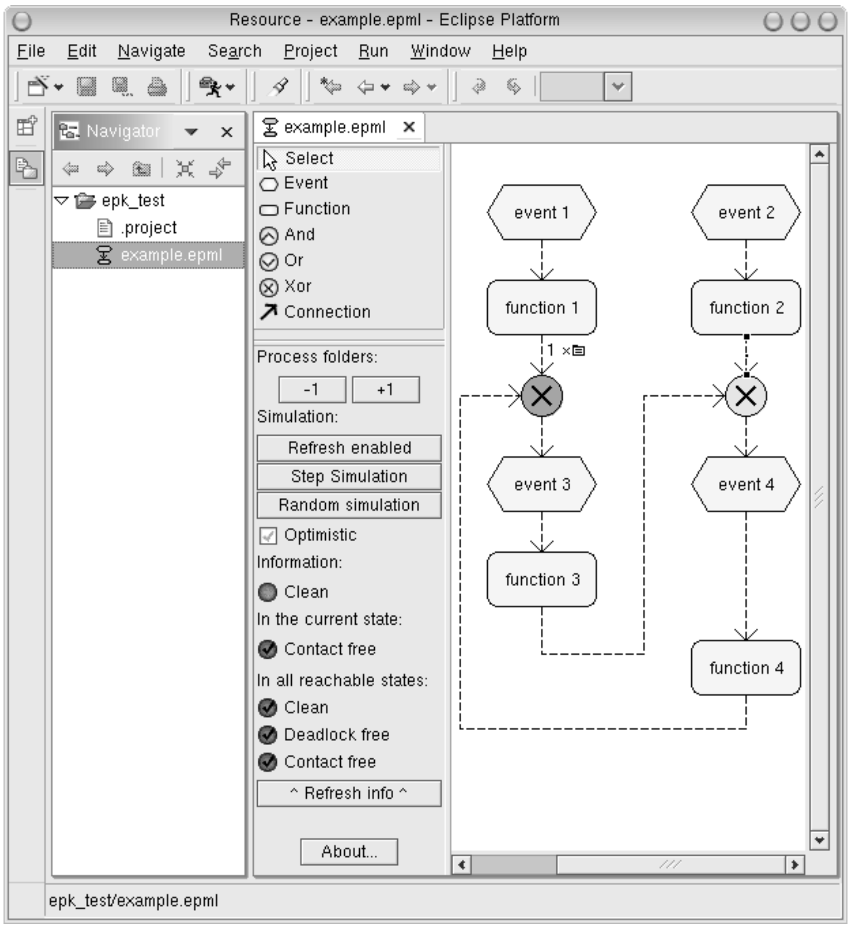


Figure 6: EPC Tools in the Eclipse environment

whether there are situations in which nodes are only blocked because of process folders on their outgoing arcs. Often, such contact situations indicate bad design.

The properties checked right now in EPC Tools, however, are quite preliminary. Once the semantics of the EPC is calculated, we could easily do much more. For example, we could check some soundness properties similar to the soundness criteria for workflow nets as proposed by van der Aalst [vdAvH02] or we could check properties by applying model checking. This should take less time than calculating the semantics. The main problem is to identify the properties that are relevant for EPCs.

Overview on the functionality The EPC Tools plugin can be used to edit, to simulate, and to analyse EPCs with the help of graphical control elements integrated into the Eclipse environment. The editor functions are provided by a tool palette containing buttons for adding nodes and arcs to the EPC. Pushing the "select" button allows a user to move, rename, and scale nodes directly by clicking on them. Some other functions like undo commands are accessible through a context menu. Figure 7 shows all these editor functions. In addition, EPC Tools provides a print function and allows a user to zoom into

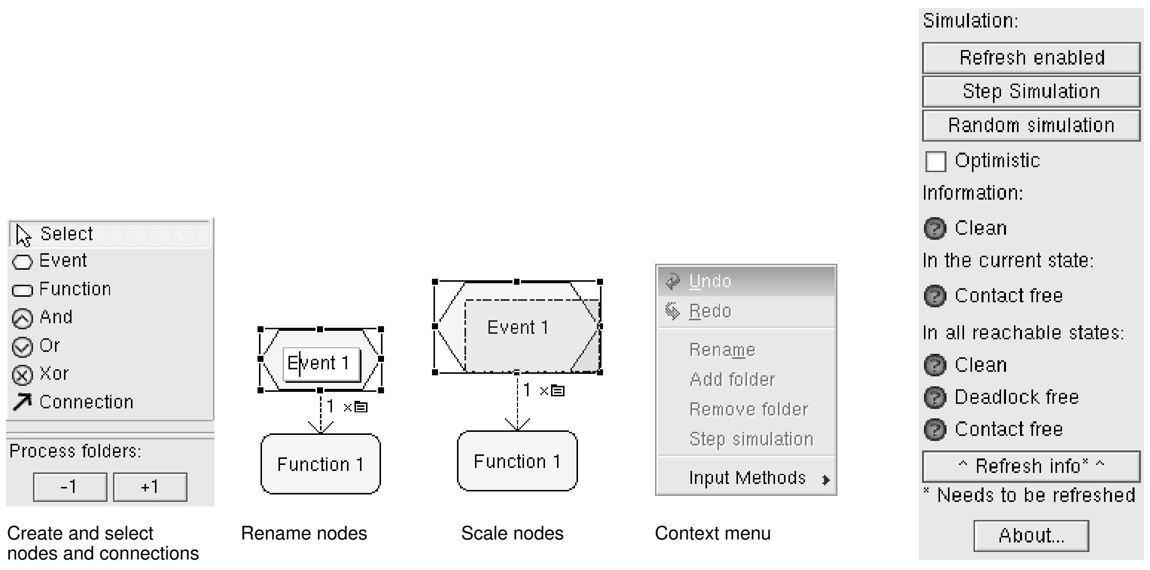


Figure 7: Editor and simulator functions

and out of EPC diagrams by using the standard Eclipse toolbar and the main menu.

The simulator functions are mainly located in the panel shown in Fig. 7 on the right side. It is possible to highlight all currently enabled nodes, and then to simulate one step by specifying a node or by randomly choosing a node. The randomized simulation can be very useful when simulating several steps consecutively by simply clicking one button. A checkbox defines whether the simulation should be done according to the optimistic or according to the pessimistic transition relation computed by the fixed-point iteration algorithm. This option is important when the simulated EPC is unclean, otherwise it does not make any difference. In the same panel, there are some LEDs corresponding to the properties of the EPC. This information can be updated by pushing the “refresh” button. Then, the LEDs light up green or red in order to indicate the valid and invalid properties.

6 Conclusion

In this paper, we have shown that the semantics of an EPC can be efficiently calculated by using ROBDDs and techniques from model checking. With the presented optimizations, the simulation of medium size EPCs works quite well and is practically feasible. Moreover, it is quite easy to adapt this algorithms to slightly different semantics by using different formulas for defining the semantics of the nodes.

The presented algorithms have been implemented in a new Tool for EPCs, which is Eclipse based and is called *EPC Tools*. This tool comes with a graphical editor and it is easy to extend it by new features. EPC Tools is open source published under the GNU Public License, which might make it a good starting point for an open source tool for EPCs. It can be obtained from [CuKi04].

Acknowledgment We would like to thank Jan Mendling for the EPML files of the examples from Fig. 5, which we used as a practical benchmark for the calculation of the semantics.

References

[vdADK02] van der Aalst, W., Desel, J., and Kindler, E.: On the semantics of EPCs: A vicious circle. In: Nüttgens, M. und Rump, F. J. (Eds.), *EPK 2002, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*. pp. 71–79. November 2002.

[vdAvH02] van der Aalst, W. and van Hee, K.: *Workflow Management: Models, Methods, and Systems*. Cooperative Information Systems. The MIT Press, 2002.

[BCM$^+$92] Burch, J., Clarke, E., McMillan, K., Dill, D., and Hwang, L.: Symbolic model checking: 10^{20} states and beyond. *Information and Computation* 98:142–170, 1992.

[Br86] Bryant, R. E.: Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers* 35(8):677–691, 1986.

[CGP99] Clarke, E., Grumberg, O., and Peled, D.: *Model checking*. MIT Press, 1999.

[Cu04] Cuntz, N.: Über die effiziente Simulation von Ereignisgesteuerten Prozessketten. Master's thesis. University of Paderborn, Department of Computer Science, June 2004.

[CuKi04] Cuntz, N. and Kindler, E.: The EPC Tools Project. http://www.upb.de/cs/kindler/research/EPCTools, 2004.

[Ecl] The Eclipse Foundation: The Eclipse platform. http://www.eclipse.org.

[HR00] Huth, M. and Ryan, M.: *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge University Press, 2000.

[Ki04a] Kindler, E.: The Model Checking in Education (MCiE) Project. http://www.upb.de/cs/kindler/teaching/MCiE, 2004.

[Ki04b] Kindler, E.: On the semantics of EPCs: Resolving the vicious circle. In: Desel, J., Pernici, B., and Weske, M. (Eds.), *Business Process Management, Second International Conference, BPM 2004*. *LNCS* 3080, pp. 82–97. Springer, June 2004. (An earlier version of this paper was presented at EPK 03.)

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

[LSW98] Langner, P., Schneider, C., and Wehler, J.: Petri Net Based Certification of Event driven Process Chains. In: Desel, J. and Silva, M. (Eds.), *Application and Theory of Petri Nets 1998*. *LNCS*1420, pp. 286–305. Springer, 1998.

[Mc93] McMillan, K. L.: *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[MN04a] Mendling, J. and Nüttgens, M.: Exchanging EPC Business Process Models with EPML. In: Mendling, J. and Nüttgens, M. (Eds.), *XML interchange formats for business process management, proceedings of the 1^{st} Workshop XML4BPM 2004*, pp. 61–80. March 2004.

[MN04b] Mendling, J. and Nüttgens, M.: Transformation of ARIS Toolset's AML to EPML. To appear.

[NR02] Nüttgens, M. and Rump, F. J.: Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In: *PROMISE 2002, Prozessorientierte Methoden und Werkzeuge fürr die Entwicklung von Informationssystemen*. *GI Lecture Notes in Informatics* P-21, pp. 64–77. Gesellschaft für Informatik, 2002.

[Ri00] Rittgen, P.: Quo vadis EPK in ARIS? *Wirtschaftsinformatik*. 42:27–35, 2000.

Transformation of ARIS Markup Language to EPML

Jan Mendling[†], Markus Nüttgens[‡]
[†]Vienna University of Economics and BA
jan.mendling@wu-wien.ac.at
[‡]Hamburg University of Economics and Politics
nuettgens@hwp-hamburg.de

Abstract: Heterogeneous and proprietary interchange formats pose a major problem for business process management. This applies in particular to processes that have been modelled as Event-Driven Process Chains (EPCs). This paper addresses heterogeneous representations of EPCs. We take ARIS Markup Language (AML)[1], the proprietary interchange format of ARIS Toolset, as a starting point and discuss its advantages and shortcomings. Afterwards, we propose a mapping from AML to EPC Markup Language (EPML) and discuss the implementation of these mappings as a XSLT transformation program. The mappings reveal that all major AML elements can be transformed to similar EPML elements. Furthermore, the XSLT program makes EPC models managed by ARIS Toolset available in EPML format.

1 Introduction

Business process modeling (BPM) is supported by various graphical modeling tools. Currently, there are at least 14 major tools for BPM available on the market [Je04]. Most of these tools use heterogeneous proprietary formats for import and export. That is in particular true for tools that support modelling of Event-Driven Process Chains (EPC). As a consequence, moving model data from one tool to another requires non-trivial transformations and detailed knowledge about the data formats of each tool involved. This is a major road block for integrating different BPM tools. A recent survey by Delphi Group identifies the lack of a commonly accepted interchange format as one of the major hindrances for BPM [De03].

This heterogeneity of proprietary data formats has been the major motivation for the definition of EPML (EPC Markup Language) [MN02, MN03, MN04a]. EPML aims to serve as a tool-neutral, XML-based *interchange format* for EPC business process models. Accordingly, it is related to other tool-neutral interchange formats like OMG's XML Metadata Interchange (XMI) [Ob03], the Petri Net Markup Language (PNML) [BCvH^{+}03] or the XML Process Definition Language (XPDL) [Wo02] proposed by the Workflow Management Coalition. Furthermore, EPML can serve as an *intermediary format* between hetero-

[1]ARIS, ARIS Toolset, and AML are trademarks of IDS Scheer AG. The use of registered names and trademarks in this paper does not imply that such names are free for general use.

geneous tools. This is especially economic when the number of tools is large. Instead of defining bilateral transformations between every pair of tools, the usage of an intermediary format reduces the number of transformations from $O(n^2)$ to $O(n)$ [WHB02]. EPML is especially suited for this purpose, because it represents EPCs in a generic, tool-neutral manner. When the interchange format is good to read and understand (for readability see [SW01, AN02, MN04b]) software developers can faster write transformations programs from and to that format, e.g. with XSLT [Cl99]. EPML has been designed to comply with the design principle of readability [MN04a] in order to allow fast and easy development of transformation programs.

The EPC method [KNS92] is very much related to ARIS (Architecture of Integrated Information Systems) [Sc00] and IDS Scheer AG as one of its major supporters. Today, many EPC business process models are maintained by the help of ARIS Toolset of IDS Scheer AG. ARIS Toolset supports import and export of business process models in a proprietary XML-based interchange format which is called ARIS Markup Language (AML) [ID03]. Although AML also builds on XML it is much more difficult to understand and to transform than data available in EPML. This is a major hindrance for the AML-based integration of ARIS Toolset with other tools and for the automated information extraction from process models available in AML. Accordingly, a transformation from cryptic AML to EPML is desirable in order to leverage the reuse to EPC model data in different applications and to make numerous EPC models available in EPML. Moreover, the transformation allows insight into the expressive power of EPML relative to AML.

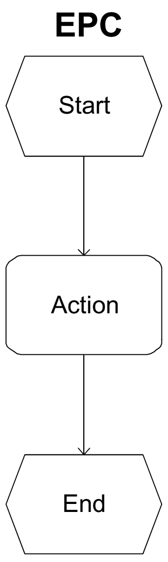
The rest of the paper is structured as follows. In Section 2 we will give an introductory example to illustrate AML and EPML. Section 3 will present and discuss AML in detail. Section 4 will introduce EPML and an extension of EPML that is capable to represent non control flow aspectsof EPC models. In Section 5 we will define transformations from AML to EPML. Furthermore, we present an XSLT program that automates these transformations. Section 6 concludes the paper and gives some outlook on future research.

2 AML and EPML by Example

Figure 1 gives the example of a very simple EPC business process model and parts of its representation both in AML and in EPML. The code wants to give a first impression of AML and EPML. Here, we only give some short explanations, details are given in the subsequent sections.

In *AML* the start event is split up in two syntax element: the object definition (`ObjDef`) represents the logical event and the object occurrence (`ObjOcc`) the appearance of the logical event in the graphical diagram. Models (`Model`) are organized in groups (`Group`) and control flow arcs are represented by logical `CxnDef` and graphical `CxnOcc` elements attached to their source object. In *EPML* the start event is captured by an `event` element, the arc is a separate `arc` element. EPC models (`epc`) are arranged in directories (`directory`). A logical representation of the start event is declared in a `definition` right after the root element. Further details on both formats will be given in the following sections.

EPC



AML

```
<AML>
<Group Group.ID="Group.Root">
   <ObjDef
      ObjDef.ID="ObjDef.1234--0-----p--"
      TypeNum="OT_EVT">
      <AttrDef
         AttrDef.ID="AttrDef.1235--0-----50l"
         AttrDef.Type="AT_NAME">
         <AttrValue>Start</AttrValue>
      </AttrDef>
      <CxnDef
         CxnDef.ID="CxnDef.1236--0-----q--"
         ToObjDef.IdRef="ObjDef.1237--0-----p--">
      </CxnDef>
   </ObjDef>
   ...
   <Model
      Model.ID="Model.1238--0-----u--"
      Model.Type="MT_EEPC">
      <ObjOcc
         ObjOcc.ID="ObjOcc.1239--0-----x-"
         ObjDef.IdRef="ObjDef.1234--0-----p-"
         SymbolNum="ST_EV">
         <Position Pos.X="0" Pos.Y="0" />
         <Size Size.dX="250" Size.dY="156" />
         <CxnOcc
            CxnOcc.ID="CxnOcc.1240--0-----y-"
            CxnDef.IdRef="CxnDef.1236--0-----q-"
            ToObjOcc.IdRef="ObjOcc.1241--0--x-">
            <Position Pos.X="125" Pos.Y="156" />
            <Position Pos.X="125" Pos.Y="312" />
         </CxnOcc>
         <AttrOcc
            AttrOcc.ID="AttrOcc.1242--0-----12"
            AttrTypeNum="AT_NAME" />
      </ObjOcc>
      ...
```

EPML

```
<epml>
<definitions>
   <definition defId="111">
      <name>Start</name>
   </definition>
   ...
</definitions>
<directory name="Group.Root">
   <epc epcId="1">
      <event id="1" defRef="111">
         <name>Start</name>
         <graphics>
            <position
               x="0" y="0"
               width="250" height="156"/>
         </graphics>
      </event>
      <arc id="14">
         <flow source="1" target="5"/>
         <graphics>
            <position x="125" y="156"/>
            <position x="125" y="312"/>
         </graphics>
      </arc>
      ...
```

Figure 1: An example of AML and EPML representation of an EPC.

3 AML Format of ARIS Toolset

AML and a so-called ARIS-Export DTD is the proprietary XML interchange format of ARIS Toolset. This section refers to the ARIS-Export.dtd and describes a subset of its syntax elements and their semantics. For a complete introduction to AML see [ID03].

An AML file starts with an `AML` element as the root element. General information like time of creation, name of the ARIS database, language, and font style is stored in subelements of `AML`. The `Group` element, also a subelement of `AML`, is a container for all model-related information. In ARIS Toolset each `Group` element refers to a directory folder of the ARIS Explorer. A `Group` must have a unique `Group.ID` attribute and it may have multiple `AttrDef`, `ObjDef`, `Model` or further `Group` subelements as children. When the `Group` and its related directory have a name, ARIS Toolset stores it in an `AttrDef` (attribute definition) subelement whose `AttrDef.Type` attribute is set to `AT_NAME`. This is typical for AML. Every specific information of objects is stored in `AttrDef` or `AttrOcc` subelements of these objects (see Figure 2).

Another principle idea of ARIS Toolset reflected in AML is the separation between definition and occurrence: each model element is first defined in an abstract way and later referenced as an occurrence in a model. This allows one logical object to be included with e.g. two occurrence in a model. Accordingly, the `Model` element contains `ObjOcc` (object occurrence) elements that refer to `ObjDef` (object definition) elements. The `ObjDef` element provides an abstract definition of an object. It has a unique `ObjDef.ID` attribute and a `TypeNum` attribute that refers to an object type, like e.g. EPC function or EPC event. Its `LinkedModels.IdRefs` attribute provides a list of ID-references to linked models.

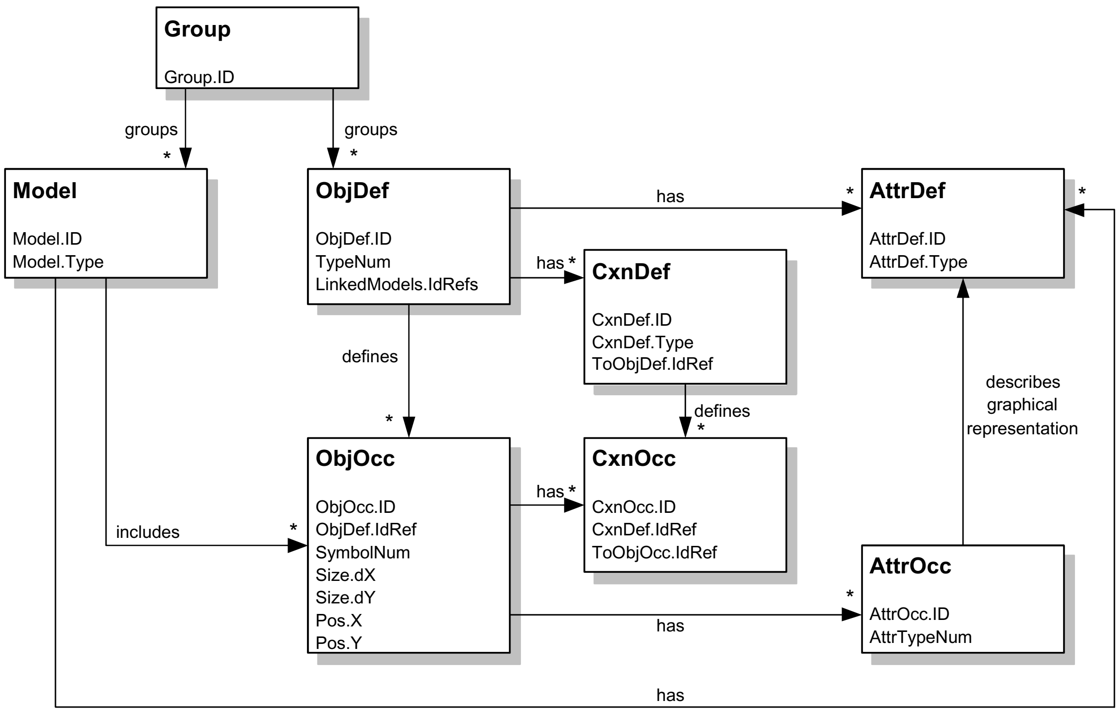


Figure 2: A UML class diagram showing a part of the AML metamodel.

These can be used e.g. for hierarchical refinement of functions. `ObjDef` elements may have multiple `AttrDef` and multiple `CxnDef` subelements. `CxnDef` elements represent arcs between objects. Each `CxnDef` has a unique `CxnDef.ID` attribute, a `CxnDef.Type` attribute, and a `ToObjDef.IdRef` attribute which represents the target of the arc. Depending on the `CxnDef.Type` attribute the arc may represent control flow, information flow, or different kinds of semantic association between the objects.

A `Model` has, among others, a unique `Model.ID` and a `Model.Type` attribute. The model type, like e.g. EPC, refers to the allowed set of objects. The `Model` element may contain `AttrDef` elements to store model specific information and `ObjOcc` elements to represent graphical elements in a visual model. An object occurrence has among others a unique `ObjOcc.ID` attribute and a reference to an object definition via the `ObjDef.IdRef` attribute. The `SymbolNum` attribute refers to a graphical icon that is used to represent the object in the visual model. An EPC function would be e.g. represented by a green rectangle with radiused edges. An `ObjOcc` element may have subelements that describe its size and its position in the visual model. Furthermore the `AttrOcc` element defines how information attached via an `AttrDef` is visually represented in a model. It has a unique `AttrOcc.ID` attribute and an `AttrTypeNum` attribute that refers to its type. This type provides a syntactical link between an `AttrOcc` and an `AttrDef` element of two associated `ObjOcc` and `ObjDef` elements. Similar to object definitions `ObjOcc` may also have multiple `CxnOcc` elements. Each of them has a unique `CxnOcc.ID` attribute and a `CxnDef.IdRef` reference to an arc definition and a reference to the target of the arc via an `ObjOcc.IdRef` attribute.

The AML metamodel is not bound to EPCs. It can represent any kind of objects whose type and icon references are understood by the application. This is a very flexible solution and allows for an easy extension of ARIS Toolset with new kinds of models and objects. Yet, there are some problems in both the design of the metamodel and its representation in AML:

Cryptic Element Names: AML uses cryptic names for some of the elements, e.g. `CxnOcc` for an arc in a model. This contradicts domain-independent XML guidelines proposed e.g. by ANSI X12 [AN02] or SWIFT [SW01] that both suggest to use telling names and no abbreviations. Such naming conventions provide for a better readability of the data and consequently for a simpler development of applications using that data.

Arc Representation: AML is to our best knowledge the only XML interchange format for BPM that uses adjacency sub-element lists representing arcs as child elements of the source node (see [MN04b]). This has some conceptual implications. Using this representation does not allow to have arcs that are not connected to a source node. Nevertheless, it could make sense to have such arcs when an incomplete model should be stored.

Separation of Definition and Occurrence: AML strictly separates abstract definition of objects and graphical representation in models. This is motivated by using one logical object multiple times in a model. As a consequence, information is split up in two XML elements while even if there is actually only one logical object. This provokes the question whether to put certain information in the definition or in the occurrence. Some design decisions of AML can be questioned in this context, e.g. it is not clear why there needs to be a `CxnDef` attached to an object definition. It would be sufficient to represent arcs in models only.

Beyond that, there is a flaw in the way how AML is used by ARIS Toolset. The types of object definitions and the icons of object occurrences are stored in the `TypeNum` and the `SymbolNum` attribute. The values of these attributes are neither enumerated in the DTD, nor do they have a telling name. An EPC function has e.g. an object type `OT_FUNC` and a symbol type `ST_FUNC`. As these predefined type values are not documented in the DTD, the developer has to find out about their meaning by analyzing AML code of process models. That fact contributes to AML's limited readability. This shortcoming does not really count if one wants to exchange models only between different ARIS Toolset implementations. But as soon as these models have to be moved to other applications or have to be used in a different context, non-trivial transformations are needed. The limited readability of AML, then, is a road block for developing transformation programs. The following section will explain the EPML representation of EPCs and an extension to capture non control flow aspects of business process models.

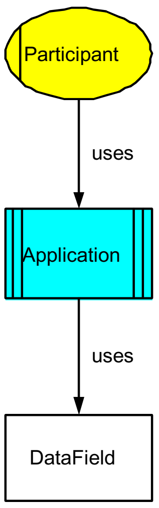
4 EPML and Non Control Flow Aspects

EPML is a XML-based tool-neutral interchange format for EPC business process models [MN02, MN03, MN04a]. The `epml` element is the root of every EPML file. It contains among others a `directory` element that can nest further directories and `epc` models. Each of these models is identified by a unique `epcId` and a `name` attribute. An `epc` element is a container for multiple control flow elements like `event`, `function`, `processInterface`, as well as `and`, `or`, and `xor` connectors, and multiple control flow `arc` elements. Each of these elements is identified by a unique `Id` attribute and a `name` element. The `function` and the `processInterface` element may include `ToProcess` elements. The latter has a `linkToEpcId` attribute representing a logical pointer to a subprocess of a function or to a subsequent process of a process interface. Each `arc` has a `flow` element whose `source` and `target` attributes represent the source and the target of the control flow arc. All EPC elements may have a `graphics` element. This element may contain `position`, `fill` (not applicable for arcs), `line`, and `font` visualization information. For control flow elements the `position` element specifies the `x` and the `y` position of the top left corner of a bounding box. Its size is indicated via the attributes `width` and `height`. Control flow arcs may have multiple `position` elements, each representing a point of a polyline. In the most simple case there are two position elements to represent the start point and the end point of the arc. For further details and further syntax elements of EPML we refer to [MN04a].

In general there are two categories of information that are frequently added to a business process model. First, *attributes* represented as *(name,value)* pairs can be used to attach statistical or configurational data to a process or to process objects. Second, various *objects* involved in the execution of a business process are frequently displayed as icons in the visual process model. Dedicated elements of EPML have been defined to represent both these kinds of information (see [MN04a]). Yet, a clear separation of textual attributes and graphical object icons like proposed by AML is also desirable for EPML in order to provide for a better tool orientation. As a consequence, we propose in the following some modifications to the way such additional information is represented in EPML.

The new Version 1.1 of the EPML Schema renames the former elements `view`, `unit`, and `unitReference` to `attributeTypes`, `attributeType` as well as `attribute` elements, respectively. Arbitrary `attributeTypes` can be declared as top-level elements of an EPML file. Single `attribute` elements can be attached to `epc` elements and to all its child elements like e.g. `function` elements. The `attributeType` may have a `description` and it must have a unique `typeName` attribute. This type name is referenced in the `typeRef` attribute of an `attribute` element. The attribute type declaration provides for a consistent naming of extension attributes used by individual tools.

Moreover, the new EPML version adds non-control flow elements which can be displayed in a graphical EPC process model. Figure 3 shows an example of these new EPC subelements. A participant uses an application and the application uses a certain data field e.g. to execute a task. The relationship is declared in a `definition` element at the top of the EPML file.



```
<definition defId="0"
     type="relationshipType">
     <name>uses</name>
</definition>
…
<epc epcId="1" name="example">
     <participant id="1">
          <name>Participant</name>
     </participant>
     <relation id="12" defRef="0"
          from="1" to="2"/>
     <application id="2">
          <name>Application</name>
     </application>
     <relation id="23" defRef="0"
          from="2" to="3"/>
     <dataField id="3">
          <name>DataField</name>
     </dataField>
```

Figure 3: Example of new EPC subelements.

In contrast to ARIS Toolset that offers about 100 different icons and object type, EPML restricts itself to three objects: `dataField`, `participant`, and `application`. These three process objects are also found in both XPDL [Wo02] and, with different names, the Architecture of Integrated Information Systems (ARIS) [Sc00]. All these three elements have a `name` and a `description` element and they are identified by a unique `id` attribute. Furthermore, they may have `graphics` and `attribute` elements. Relationships between these elements or between these elements and control flow elements are represented by `relation` elements. A `relation` is a directed edge between an element whose `id` is referenced in the `from` attribute and another element referred to in the `to` attribute. The `relation` element is related to the `arc` element. Yet, the syntactical distinction between both allows to easily identify control flow with `arc` elements. The `relation` elements may have multiple `graphics` elements. They can also contain `attribute` elements. Furthermore, the `defRef` attribute of a relation must reference a `defId` of a `definition` element. The definition is meant to describe the semantics of the relationship. Including these new elements an `epc` may now have `function`, `event`, `processInterface`, `and`, `or`, `xor`, and `arc` elements as well as `participant`, `application`, `dataField`, and `relation` elements as children.

5 Transformation from AML to EPML

In this section we will present the transformation from AML to EPML. This transformation has been implemented as an XSLT program. For further information see *http://wi.wu-wien.ac.at/~mendling/EPML*. In the following we will illustrate the transformation subdivided into five aspects: generation of positive integer identifiers, generation of EPML header fields, mapping of navigation structure, mapping model elements, and mapping model element subelements (see Table 1). To avoid confusion we use the namespace prefixes `aml`, `epml`, and `xsl` in the text.

Table 1: Mapping of AML elements to EPML elements

AML	EPML
`aml:Model.ID`	`epml:epcId`
`aml:ObjDef.Id`	`epml:defId`
`aml:ObjOcc.Id`	`epml:id`
`aml:AttrDef.Type`	`epml:defId`
`aml:ObjDef`	`epml:definition`
`aml:AttrDef`	`epml:attributeType`
`aml:Group`	`epml:directory`
`aml:Model`	`epml:epc`
`aml:ObjOcc`	different EPML elements
`aml:CxnOcc`	`epml:arc` or `epml:relation`
`aml:Pos.X`	`epml:x`
`aml:Pos.Y`	`epml:y`
`aml:Size.dX`	`epml:width`
`aml:Size.dY`	`epml:height`

EPML requires identifying attributes to have positive integer values. As AML identifiers are of string type there has to be a mapping. Figure 4 illustrates how a list of *(identifier,position)* pairs can be generated. An XSLT node set of e.g. all `aml:ObjOcc` is processed via a `xsl:for-each` loop. Each pair of an identifier and its position in the node set is written to the list. Later in the transformation program this list can be queried via a `xsl:select` of an XSLT `xsl:value-of` statement (see Figure 4).

```
<xsl:variable name="ObjOccTable">
     <xsl:for-each select="//*[name()='ObjOcc']">
          <xsl:value-of select="@ObjOcc.ID" />
          <xsl:value-of select="concat(' ',position(),' ')" />
     </xsl:for-each>
</xsl:variable>
```

```
<xsl:value-of select="substring-before(substring-after(
     substring-after($ObjOccTable,$Id),' '),' ')" />
```

Figure 4: A list of ObjOcc.ID-Position pairs and a related query.

EPML header fields include definition and attribute type elements. First, if there are two or more `aml:ObjOcc` elements that share an `aml:OccDef` element a corresponding definition element has to be included to represent such a relationship. Subelements of EPC models have to reference the `epml:defId` attribute of a definition in their `epml:defRef` attribute. Second, definition elements have to be added for non control flow relationships. Accordingly, `epml:relation` elements reference the respective `epml:defId` in their `epml:defRef` attribute. Third, attribute type elements have to be added for all types of attributes used in the EPC models. These attributes have to reference the correct `epml:typeId` in their `epml:typeRef` attribute. For all these elements and their identifiers dedicated *(identifier,position)* lists have to be stored in variables similar as described above for `aml:ObjOcc.ID` attributes.

The navigation structure of AML and EPML is very similar. In AML the `aml:group` element corresponds to the `epml:directory` element. Moreover, the `aml:model` element is mapped to an `epml:epc` element. The `aml:Model.ID` attribute maps to an `epml:epcId` attribute following the mechanism described above concerning identifiers. Figure 5 illustrates how the name of a directory or a model is retrieved from an AML element. It is stored in the `aml:AttrValue` element of an `aml:AttrDef`. The corresponding `aml:AttrDef.Type` is AT_NAME.

```
<xsl:attribute name="name">
     <xsl:value-of select="./*[name()='AttrDef']/
          *[name()='AttrValue'][../@AttrDef.Type='AT_NAME']" />
</xsl:attribute>
```

Figure 5: Mapping names from AML to EPML.

The subelements of `epml:epc` are derived from `aml:ObjOcc` and `aml:CxnOcc` elements. The type of `aml:ObjOcc` elements can be identified via its `aml:SymbolNum` attribute and the `aml:TypeNum` attribute of its corresponding `aml:ObjDef` element. Table 2 gives an overview of the mapping rules. For the `aml:CxnOcc` element two mappings have to be distinguished. First, if the source and the target of the edge are both EPC control flow elements, then the edge maps to an `epml:arc` element. Second, if at least the source or the target of the edge is not an EPC control flow element, then the edge maps to an `epml:relation` element.

The different subelements of `epml:epc` may contains further information. We use the case of an `epml:function` to explain the mappings. Similar to identifiers and names of directories as described above the `id` attribute and the `name` element of a function can be generated from the AML elements. Moreover, the `graphics` element of a function includes position information. The corresponding `epml:x`, `epml:y`, `epml:width`, and `epml:height` attributes can be generated using `aml:Pos.X`, `aml:Pos.Y`, `aml:Size.dX`, and `aml:Size.dY` attributes, respectively. Linked EPC business process models captured in the `epml:linkToEpcId` attribute can be extracted from `aml:LinkedModels.IdRefs` attributes. Finally, `aml:AttrDef` and `aml:AttrOcc` elements map to `epml:attribute` elements.

The proposed mapping from AML to EPML permits the following conclusions. First, EPML is capable to capture the essential AML concepts without loss of information. This has been demonstrated by the implementation of the described XSLT transformation program. Second, EPML uses a more intuitive representation of EPCs than AML. This rather subjective statement is supported by two facts: EPML uses telling element names inspired

Table 2: XSLT-Mapping of AML's `ObjOcc` elements to EPML elements

AML's `ObjOcc`	EPML
`aml:TypeNum='OT_FUNC' and aml:SymbolNum!='ST_PRCS_IF'`	`function`
`aml:TypeNum='OT_FUNC' and aml:SymbolNum='ST_PRCS_IF'`	`processInterface`
`aml:TypeNum='OT_EVT'`	`event`
`aml:TypeNum='ST_OPR_XOR_1'`	`xor`
`aml:TypeNum='ST_OPR_AND_1'`	`and`
`aml:TypeNum='ST_OPR_OR_1'`	`or`
`contains(aml:TypeNum,'_ORG') or contains(aml:TypeNum,'_PERS') contains(aml:TypeNum,'_EMPL')`	`participant`
`contains(aml:TypeNum,'_APP') or contains(aml:TypeNum,'_CMP') or contains(aml:TypeNum,'_MOD') or contains(aml:TypeNum,'_PACK')`	`application`
`contains(aml:TypeNum,'_CLS') or contains(aml:TypeNum,'_INFO') or contains(aml:TypeNum,'_KPI') or contains(aml:TypeNum,'_LST') or contains(aml:TypeNum,'_OBJ') or contains(aml:TypeNum,'_TERM')`	`dataField`

by the generic names of EPC syntax elements. Furthermore, EPC models transformed to EPML are less than half as large as the original AML files. This intuitive and generic representation of EPCs in EPML contributes to its readability. As a consequence, building EPML-based applications is easier than relying on AML. We estimate that thoroughly understanding and deciphering AML's names and abbreviations takes at least half a week, considering that AML object types and symbol types are not documented in the AML manual. Third, the transformation program makes available EPC models managed by ARIS Toolset for EPML-based applications, like EPC Tools [CK04] or EPML2SVG [MBN04].

6 Conclusion and Future Work

In this paper we presented an approach to transform EPC business process models available as files conforming to the ARIS Markup Language (AML) to EPC Markup Language (EPML). This transformation has been implemented as an XSLT program (see *http://wi.wu-wien.ac.at/~mendling/EPML*). Such a transformation is on the one hand important from a pragmatic point of view because many EPC business process models are managed by the help of ARIS Toolset and a transformation program makes these models available for EPML-based applications, like EPC Tools [CK04] or EPML2SVG [MBN04]. On the other hand the transformation offers insight into the different representational alternatives for EPCs in XML. EPML uses a more generic and more readable representation for EPCs than AML. This is helpful when building EPML-based applications. Beyond its readability, EPML is still capable to capture all essential model elements that are included in AML. Future research will be dedicated to the analysis of XML-based representation of EPCs in further business process management tools. This will include the development of related transformation programs to leverage EPML as a tool-neutral interchange format for EPCs.

Acknowledgement. The authors would like to thank the anonymous reviewers for there comments on an earlier version of this paper, which helped to improve the presentation of the ideas.

Disclaimer. We, the authors and the associated institutions, assume no legal liability or responsibility for the accuracy and completeness of any information about ARIS Toolset and AML contained in this paper. However, we made all possible efforts to ensure that the results presented are, to the best of our knowledge, up-to-date and correct.

References

[AN02] ANSI X12: ASC X12 Reference Model for XML Design. Technical Report Type II - ASC X12C/TG3/2002-xxx. ANSI ASC X12C Communications and Controls Subcommittee. July 2002.

[BCvH 03] Billington, J., Christensen, S., van Hee, K. E., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., and Weber, M.: The Petri Net Markup Language: Concepts, Technology, and Tools. In: W. M. P. van der Aalst and E. Best (eds.), *Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN 2003, Eindhoven, The Netherlands*. volume 2679 of *Lecture Notes in Computer Science*. pages 483–505. 2003.

[CK04] Cuntz, N. and Kindler, E.: On the semantics of EPCs: Efficient calculation and simulation. In: M. Nüttgens and F. J. Rump (eds.), *Proceedings of the 3rd GI Workshop on Business Process Management with Event-Driven Process Chains (EPK 2004), Luxembourg, Luxembourg*. 2004.

[Cl99] Clark, J.: XSL Transformations (XSLT) Version 1.0. W3C Recommendation 16 November. World Wide Web Consortium. 1999.

[De03] Delphi Group: *BPM 2003 – Market Milestone Report, White Paper*. 2003.

[ID03] IDS Scheer AG: *XML-Export und -Import (ARIS 6 Collaborative Suite Version 6.2 Schnittstellenbeschreibung)*. ftp://ftp.ids-scheer.de/pub/ARIS/HELPDESK/EXPORT/. Juni 2003.

[Je04] Jenz und Partner (ed.): *Business Process Modeling Tools, accessed on 3th July 2004*. http://www.jenzundpartner.de/Resources/Product_Watchlist/product_watchlist.htm. 2004.

[KNS92] Keller, G., Nüttgens, M., and Scheer, A. W.: Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Technical Report 89. Institut für Wirtschaftsinformatik Saarbrücken. Saarbrücken, Germany. 1992.

[MBN04] Mendling, J., Brabenetz, A., and Neumann, G.: Generating SVG Graphics from EPML Processes. In: M. Nüttgens and F. J. Rump (eds.), *Proceedings of the 3rd GI Workshop on Business Process Management with Event-Driven Process Chains (EPK 2004), Luxembourg, Luxembourg.* 2004.

[MN02] Mendling, J. and Nüttgens, M.: Event-Driven-Process-Chain-Markup-Language (EPML): Anforderungen zur Definition eines XML-Schemas für Ereignisgesteuerte Prozessketten (EPK). In: M. Nüttgens and F. J. Rump (eds.), *Proc. of the 1st GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2002), Trier, Germany.* pages 87–93. 2002.

[MN03] Mendling, J. and Nüttgens, M.: XML-basierte Geschäftsprozessmodellierung. In: W. Uhr and W. Esswein and E. Schoop (eds.), *Proc. of Wirtschaftsinformatik 2003 / Band II, Dresden, Germany.* pages 161 –180. 2003.

[MN04a] Mendling, J. and Nüttgens, M.: Exchanging EPC Business Process Models with EPML. In: J. Mendling and M. Nüttgens (eds.), *Proc. of the 1st GI-Workshop XML4BPM - XML Interchange Formats for Business Process Management, Marburg, Germany, March, 2004.* pages 61–79. 2004.

[MN04b] Mendling, J. and Nüttgens, M.: XML-based Reference Modelling: Foundations of an EPC Markup Language. In: J. Becker and P. Delfmann (eds.), *Referenzmodellierung - Proceedings of the 8th GI-Workshop on Reference Modelling, MKWI Essen, Germany.* pages 51–71. 2004.

[Ob03] Object Management Group: XML Metadata Interchange (XMI). Specification, Version 2.0. Object Management Group. May 2003.

[Sc00] Scheer, A. W.: *ARIS business process modelling.* Springer Verlag. 2000.

[SW01] SWIFT: SWIFT Standards XML Design Rules Version 2.3. Technical Specification (http://xml.coverpages.org/EBTWG-SWIFTStandards-XML200110.pdf). Society for Worldwide Interbank Financial Telecommunication. 2001.

[WHB02] Wüstner, E., Hotzel, T., and Buxmann, P.: Converting Business Documents: A Classification of Problems and Solutions using XML/XSLT. In: *Proceedings of the 4th International Workshop on Advanced Issues of E-Commerce and Web-based Systems (WECWIS).* pages 61–68. 2002.

[Wo02] Workflow Management Coalition: Workflow Process Definition Interface – XML Process Definition Language. Document Number WFMC-TC-1025, October 25, 2002, Version 1.0. Workflow Management Coalition. 2002.

EPK-Referenzmodelle für Verwaltungsverfahren

Oliver Thomas, Christian Seel (Dipl.-Kfm.),
Christian Seel (MScIS), Bettina Kaffai, Gunnar Martin

Institut für Wirtschaftsinformatik (IWi)
im Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI)
Stuhlsatzenhausweg 3, Geb. 43.8, 66123 Saarbrücken
[thomas|c.seel|ch.seel|kaffai|martin]@iwi.uni-sb.de

Abstract: Der Einsatz von Referenzprozessmodellen eröffnet eine Perspektive für die Prozessgestaltung in der öffentlichen Verwaltung. Allerdings erfordert dabei das komplexe Geflecht rechtlicher Vorschriften und Vorgaben sowie juristischer Ermessensspielräume für die Erstellung adäquater Modelle eine sorgfältige Prüfung und Selektion sowohl des zur Konstruktion von Referenzprozessmodellen erforderlichen Wissens als auch der zu verwendenden Modellierungssprache. Der vorliegende Beitrag zeigt am Fallbeispiel „Planfeststellungsverfahren" auf, wie Referenzmodelle unter Verwendung Ereignisgesteuerter Prozessketten für typische Verfahren der öffentlichen Verwaltung erstellt werden können und welche neuen Herausforderungen sich an die Konstruktion von Modellen in der Domäne der öffentlichen Verwaltung ergeben.[1]

1 Ausgangssituation und Problemstellung

Die mit Hilfe des Einsatzes moderner Informations- und Kommunikationstechnik (IuK) realisierbaren Effektivitäts- und Effizienzsteigerungspotenziale für die Leistungserstellung öffentlicher Verwaltungen werden seit geraumer Zeit diskutiert [Krau83; Rein86; Lenk99]. Der Ansatz des Electronic Government – kurz: eGovernment – im Sinne der „Abwicklung geschäftlicher Prozesse im Zusammenhang mit Regieren und Verwalten (Government) mit Hilfe von Informations- und Kommunikationstechniken über elektronische Medien" [vLRe01, S. 1] institutionalisiert seit Ende der 1990er-Jahre den Einsatz moderner IuK in der öffentlichen Verwaltung.

Das Erreichen zentraler Ziele des eGovernment, wie z. B. die Qualitätsverbesserung von öffentlichen Dienstleistungen oder Produktivitätssteigerungen, setzt dabei neben der rein technologischen Umsetzung vor allem die Berücksichtigung *organisationaler Gestaltungsmaßnahmen* voraus [ScSc04, S. 4 ff.]. Insbesondere die Reorganisation von Verwal-

[1] Der vorliegende Beitrag resultiert aus den Forschungsprojekten „Referenzarchitektur für E-Government" (RAFEG) sowie „Referenzmodellierung 2006 – Softwaremodelle für kleine und mittlere Softwareunternehmen durch adaptive, komponentenorientierte Referenzmodellierung: Konzept, Modellierungstechnik und Werkzeug" (RefMod06), gefördert vom Bundesministerium für Bildung und Forschung (BMBF) (Förderkennzeichen: 01 ISC 07 B und 01 IS C05 B).

tungsabläufen ist von zentraler Bedeutung. Im Sinne eines „Verwaltungs-Engineering“ sind in Abstimmung mit den strategischen Zielen organisationsübergreifende und durchgängige Prozesse zu schaffen [Gi00, S. 25f.]. Um komplexe Sachverhalte zu erheben, diese strukturiert darzustellen sowie durchgängige Prozesse zu konstruieren, hat sich in der Wirtschaftsinformatik die Verwendung von Modellen als Gestaltungsinstrument etabliert.

Die Notwendigkeit der adäquaten Gestaltung relevanter Prozessstrukturen als Basis des effektiven IuK-Einsatzes wird dabei in der Verwaltungspraxis zunehmend akzeptiert. So bildet etwa für die meisten deutschen Landesverwaltungen die Prozessoptimierung einen expliziten Bestandteil der Reform- bzw. eGovernment-Initiativen [z.B. Mauc99, S. 37ff.]. Gleichzeitig wird von den Verantwortlichen aber – neben knappen finanziellen Ressourcen, fehlenden Prozessdokumentationen oder Widerständen innerhalb der Organisation – mangelndes Know-how als Barriere für die Realisierung von Restrukturierungsmaßnahmen genannt [SeGü02, S. 109]. Zwar stellen die wissenschaftlichen Disziplinen der Betriebswirtschaftslehre und der Wirtschaftsinformatik ein umfassendes Wissen zum Prozessmanagement zur Verfügung. Für die Verwaltungsdomäne aber ist deren Adaption und praktische Anwendung bislang nur punktuell erfolgt [Sche02; Bec+03]. Ein integraler Bestandteil der weitaus meisten Verfahren in den genannten Wissenschaftsdisziplinen ist die Konstruktion von Modellen.

Mit dem Arbeitsgebiet der Referenzmodellierung steht ein viel versprechender Ansatz zur Erleichterung der Konstruktion spezifischer Modelle zur Verfügung. *Referenzmodelle* werden als spezielle Informationsmodelle verstanden, die zur Unterstützung der Konstruktion anderer Informationsmodelle genutzt werden [Hars94; Sche97; Schü98; vBro03]. Die Verwendung von Referenzmodellen beschleunigt die Konstruktion spezifischer Modelle, erhöht deren Qualität, da auf erprobtes Domänenwissen, das den Best- bzw. Common-Practice darstellt, zurückgegriffen wird. Außerdem wird so unterschiedlichen Referenzmodellnutzern eine einheitliche Kommunikation über die im Referenzmodell dargestellten Sachverhalte ermöglicht. Der Referenzmodellnutzer hat bei der Referenzmodellverwendung die Aufgabe, eine *Referenzmodelladaption* vorzunehmen, die durch informationstechnische Werkzeuge unterstützt werden kann. Die mit diesem Begriff charakterisierte Ableitung spezifischer Modelle aus einem Referenzmodell entspricht im übertragenen Sinne der Bildung von Varianten des Referenzmodells [Schü98; Schl00; Sche02; vBro03].

Im Folgenden wird die Erstellung von Referenzprozessmodellen für typische Verwaltungsabläufe mit Hilfe der Ereignisgesteuerten Prozesskette (EPK) dargestellt. Die Eignung der EPK wird dabei zunächst im Kontext der Domänencharakteristika dargelegt (Abschnitt 2). Die konkrete Anwendung auf den Bereich der Planfeststellungsverfahren zeigt dann die Vorgehensweise und insbesondere die aus den Verwaltungsspezifika resultierenden Herausforderungen für die Modellierung (Abschnitt 3). Die Ausführungen schließen mit einer Betrachtung des Handlungsbedarfs und insbesondere der Forderung nach Handlungsanleitungen für die Konstruktion von EPK-Referenzmodellen für Verwaltungsverfahren (Abschnitt 4).

2 Referenzprozessmodellierung in der öffentlichen Verwaltung mit Ereignisgesteuerten Prozessketten

2.1 Öffentliche Verwaltung und Referenzprozessmodellierung

Gemeinhin wird die öffentliche Verwaltung – ausgehend vom System der Gewaltenteilung – als die Grundgesamtheit staatlicher Tätigkeiten umschrieben, die nicht zur Gesetzgebung, Rechtsprechung und Regierung gehören [z.B. Thie84, S.2; Reic87, S.3]. Positiv formulierte Ansätze beschreiben die Verwaltung als Vollzugsorgan, das von der Legislativen durch Rahmenzielsetzungen gesteuert und durch die Judikative kontrolliert wird [z.B. Reic87, S.3]. Eine weitergehende Charakterisierung kann anhand der Aufgaben, der Leistungen und der Organisationsform vorgenommen werden.

Die *Aufgabenstellung* charakterisiert die Verwaltung als Organisation, die öffentliche Aufgaben erledigt. Diese leiten sich aus den öffentlichen, gemeinwohlorientierten Zielen ab, die sich wiederum an den staatlichen Zielen orientieren. Als deren Erfüllungsorgan konkretisieren die Verwaltungsinstitutionen politische Entscheidungen, fordern zu bestimmten Handlungen auf oder führen Transaktionen durch. Die Rahmenparameter werden dabei durch die politischen Entscheidungen definiert, die wiederum in Rechtsnormen Ausdruck finden [Lenk99, S.7]. Das Recht bildet somit die Grundlage des Verwaltungshandelns, dementsprechend sind viele Leistungserstellungsprozesse durch Rechtsvorgaben mehr oder weniger intensiv strukturiert. Die rechtliche Bindung bezieht sich dabei nicht nur auf die Abläufe per se, sondern auch auf die relevanten Dokumente und die Entscheidungsfindung [WiTL01, S.437f.].

Bei den *Leistungen* als Ergebnis der Aufgabenerfüllung handelt es sich dabei bis auf wenige Ausnahmen, wie etwa die Erstellung von Sachleistungen oder die Erbringung von körperlichen Dienstleistungen an Menschen, ausschließlich um eine Informationstätigkeit [Lenk99, S.7]. Dementsprechend determinieren die Faktoren Information und Wissen einen Großteil der Verwaltungsprozesse. Für die Behörden als „Produzent" von informationellem Output bilden Informationen gleichsam den zentralen „Rohstoff" der Leistungserstellung. Von Relevanz für das Verwaltungshandeln sind dabei insbesondere Informationen über die Umwelt, über das eigene Handeln und über die zu befolgenden Regeln [Lenk99, S.13]. Das daraus generierte Wissen hat gemeinsam mit dem Dienstwissen der Mitarbeiter entscheidenden Einfluss auf das jeweilige Verwaltungshandeln. Für die Bearbeiter mit Ermessensbefugnissen bildet die Kenntnis der rechtlichen Vorgaben, des relevanten Umfelds, der Handlungskonsequenzen und nicht zuletzt der Einflussmöglichkeiten und der Effektivität des eigenen Handlungsinstrumentariums die Voraussetzung zur Aufgabenerfüllung [WiTL01, S.438f.].

Die *Organisationsstruktur* zur Leistungserbringung wird in deutschen Verwaltungen durch das Anfang des letzten Jahrhunderts formulierte Bürokratiemodell von WEBER geprägt [z.B. Mayn68, S.27f.; ScPr03, S.16]. Neben der rechtlichen Bindung des Verwaltungshandelns sind hier die funktional strukturierte Aufbauorganisation, die klar abgegrenzten Zuständigkeiten und Kompetenzen, die feste Amtshierarchie und die Aktenmäßigkeit der Amtsführung als Charakteristika zu nennen. Trotz dieser auf den ersten Blick als sehr restriktiv erscheinenden Grundgesamtheit von Bestimmungsfaktoren resultiert keineswegs ein einheitlicher Typ von stark zentralisierten, formalisierten und standardisierten Geschäftsprozessen. Es wird vielmehr ein Rahmen definiert, innerhalb dessen gemäß der jeweiligen Aufgabe bzw. des Einzelfalls das Verwaltungshandeln mittels Ermessen oder Konsensbildung gestaltet werden kann. Die relevanten Steuerungsmechanismen können gemäß ihrer rechtlichen Verbindlichkeit kategorisiert werden. So unterscheidet bspw. die Prozesssteuerung durch legislative Programmierung zweck- und konditionalprogrammiertes Verwaltungshandeln. Der Freiheitsgrad bei der Ausführung einer Maßnahme reicht dabei von der Finalprogrammierung, bei der lediglich Ziele vorgegeben, bis hin zu vollprogrammierten Entscheidungen, bei denen alle Entscheidungsbedingungen exakt definiert und vorgegeben werden [Enge96].

Aus der Perspektive des eGovernment bildet dabei die Schaffung adäquater Gesamtprozessszenarien unter Berücksichtigung aller relevanten Anspruchsgruppen die Grundvoraussetzung für die Realisierung der Nutzenpotenziale des IuK-Einsatzes. „Produktivitätssteigerung mittels IKT heißt Reorganisieren, heißt in Besitzstände eingreifen, heißt also ‚Führungsschwerstarbeit'." [Rein95, S. 129] Eine grobe Kategorisierung von Verwaltungsprozessen kann durch eine Unterscheidung von Hilfsprozessen zur Beschaffung und Pflege der Verwaltungsressourcen und Prozessen des operativen Verwaltungshandelns vorgenommen werden. Letztere bilden das Kernspektrum der Verwaltung und reichen von wohldefinierten Prozessen bis hin zu komplexen, durch rechtliche Fragen und Entscheidungen dominierten Abläufen [Gi00, S. 18]. Gerade die große qualitative Bedeutung der letztgenannten Kategorie muss im Rahmen der Restrukturierung explizit berücksichtigt werden [Gi00, S. 298; WiTr02]. Die weiteren Ausführungen nehmen daher Bezug auf diese Prozesstypen.

Der Einsatz von Referenzprozessmodellen wird durch die aus dem beschriebenen rechtlichen und bürokratischen Rahmen resultierenden Strukturanalogien begünstigt, wobei jedoch eine Differenzierung von „individueller Situation und wiederkehrenden Mustern (Struktur, Modelle) in Verwaltungsprozessen" [WiTr02] vorgenommen werden muss. Die Verwendung derartiger Modelle kann die Komplexität von eGovernment-Projekten erheblich reduzieren und die Umsetzung vereinfachen. So wird etwa im Rahmen der Soll-Modellierung der Designprozess durch den Einsatz von Referenzmodellen erheblich beschleunigt. Außerdem partizipiert die Verwaltung an dem im Modell gespeicherten Erfahrungswissen. Sie wird in die Lage versetzt, soweit wie möglich Standardprozesse zu übernehmen und sich vornehmlich auf die Bereiche zu konzentrieren, die eine individuelle Lösung erfordern [BeAN04; Heib04]. Für die Modellkonstruktion und -verwendung ist dabei die Nutzung einer adäquaten Modellierungssprache von essenzieller Bedeutung. Im Folgenden wird daher die Eignung der Ereignisgesteuerten Prozesskette für die Modellierung von Verwaltungsprozessen dargelegt.

2.2 Eignung der EPK als Modellierungssprache für Referenzprozessmodelle in der öffentlichen Verwaltung

Insbesondere im deutschsprachigen Raum hat sich die EPK zur Konstruktion von Referenzprozessmodellen auf fachkonzeptioneller Ebene etabliert und – nicht zuletzt aufgrund ihrer Anwendungsorientierung und einer umfassenden Werkzeugunterstützung – einen hohen Grad der Verbreitung und Akzeptanz in der Praxis gefunden [Nütt97; Ritt00; NüRu02, S. 64]. Für das Verwaltungsumfeld hat sich die Eignung der EPK bereits in der Vergangenheit im Rahmen konkreter Projekte bewiesen [ScNZ96, S. 14 ff.; BeAN03a, S. 36 ff.].

Die Grundelemente der EPK sind Ereignisse, Funktionen, Kontrollflusskanten und Verknüpfungsoperatoren [KeNS92]. Ferner resultieren aus der Ableitung der EPK als zentrale Modellierungssprache der Architektur integrierter Informationssysteme (ARIS) erweiterte Aussagen, die auf dem ARIS-Sichtenkonzept aufbauen. Diese Aussagen werden durch Annotation von zusätzlichen Sprachkonstrukten an EPK-Funktionen getroffen. SCHEER [Sche02] schlägt bspw. Sprachkonstrukte vor, die Umfelddaten, Nachrichten, menschliche Arbeitsleistung, maschinelle Ressourcen und Computer-Hardware, Anwendungssoftware, Leistungen in Form von Sach-, Dienst- und Informationsdienstleistungen, Finanzmittel, Organisationseinheiten oder Unternehmungsziele repräsentieren. Die Verbindung der Konstrukte mit EPK-Funktionen wird über Kanten hergestellt, die – neben dem Kontrollfluss – unter anderem in Organisations-/Ressourcen-, Informations-, Informationsdienstleistungs- und Sachleistungs- sowie Finanzmittelfluss unterschieden werden [Sche02].

Den spezifischen, aus den oben skizzierten Charakteristika der Anwendungsdomäne resultierenden Anforderungen kann durch diese Grundgesamtheit an Sprachkonstrukten weitestgehend Rechnung getragen werden. Zunächst ist hier die aus der bürokratischen Organisation resultierende Kompetenzverteilung zu nennen, die nicht nur die Abbildung der relevanten Organisationseinheiten in den Modellen erfordert, sondern insbesondere auch ein Vorhalten wohldefinierter Schnittstellen notwendig macht. In diesem Zusammenhang sind auch externe Anspruchsgruppen, wie z. B. Bürger oder Interessensgemeinschaften, zu berücksichtigen. Der informationsverarbeitende Charakter der Verwaltungsabläufe bedingt ferner die Darstellung der zugrunde liegenden Informationen sowie der unterstützenden Systeme. Für die Prozessoptimierung ist weiterhin das Einbeziehen von Mitarbeitern der involvierten Fachbereiche und deren Fach- und Dienstwissen etwa bei der Definition von Verbesserungsmaßnahmen von essenzieller Bedeutung. Zur Sicherstellung der Handhabbarkeit in Restrukturierungsprojekten muss daher die Verständlichkeit und Nachvollziehbarkeit der Referenzmodelle auch für „Modellierungslaien“ gewährleistet sein. Die potenziell hohe Anzahl von Nutzern lässt es hier außerdem als sinnvoll erscheinen, eine Multiperspektivität in der Modelldarstellung zu realisieren. Die Verwendung der Referenzprozessmodelle in verschiedenen Verwaltungen setzt schließlich eine einfache Vergleichbarkeit verschiedener Modelle voraus [BeAN03a, S. 35 f.; BeAN03b, S. 862 ff.].

Zusammenfassend kann konstatiert werden, dass vor dem Hintergrund dieser Anforderungen der Einsatz der EPK zur Referenzmodellierung in der öffentlichen Verwaltung angemessen scheint. Bei der Erstellung eines Referenzmodells ist sowohl die Einbeziehung von Fachexperten der betrachteten Domäne, in diesem Fall der Verwaltung, als auch die Einbeziehung von Methodenexperten erforderlich. Die hohe Anschaulichkeit der EPK-Modelle sichert das Verständnis und die Nachvollziehbarkeit bei den Fachexperten in den betroffenen Behörden, erlaubt aber ebenso eine aus methodischer Sicht adäquate Darstellung. Andere Modellierungssprachen wie UML oder Petri-Netze, die grundsätzlich auch geeignet wären entsprechende Prozesse darzustellen, sind für Fachexperten aufgrund ihres Formalisierungsgrads weniger anschaulich und verständlich, wie die Erfahrungen der Autoren bei Modellierung von Verwaltungsprozessen im Regierungspräsidium Leipzig gezeigt haben. Des Weiteren enthalten erweiterte EPK-Modelle Konstrukte, wie Dokumente, oder Teile von Organigrammen, die für die Verwaltung wertvolles Know-how darstellen, aber in den anderen beiden genannten Modellierungssprachen nicht vorhanden sind. Darüber hinaus sprechen die weite Verbreitung und der hohe Bekanntheitsgrad der EPK, z. B. beim Customizing von ERP-Systemen, ihre Tool-Unterstützung sowie die Möglichkeit der Komposition/ Dekomposition für den Einsatz von EPK-Modellen. Der Umfang der sprachlichen Konstrukte ermöglicht die umfassende Abbildung relevanter Verwaltungsszenarien. In der Praxis werden dabei komplexe Szenarien, wie sie im Umfeld der Behörden zu finden sind, aus Gründen der Übersichtlichkeit in der Regel nicht durch ein einziges EPK-Modell repräsentiert, sondern durch mehrere miteinander verbundene und dekomponierte Teilmodelle. Deren Verknüpfung wird dabei über so genannte Prozesswegweiser [KeTe96] erreicht. Funktionsverfeinerungen dienen der Hierarchisierung von EPK-Modellen [Nütt95]. Am Beispiel des Planfeststellungsverfahrens wird im Folgenden ein derartiges EPK-Referenzmodell für Verwaltungsabläufe mit hohen Freiheitsgraden vorgestellt.

3 Konstruktion von EPK-Verwaltungsverfahrensmodellen am Beispiel der Planfeststellung

3.1 Das Planfeststellungsverfahren

Planfeststellungsverfahren (PFV) sind ein Grundtyp der Verwaltungsverfahrensarten und zeichnen sich durch eine starke Formalisierung aus, die sich bspw. in klaren Vorgaben zur Öffentlichkeitswirkung oder Verhandlungsabläufen äußert. Dies bedeutet jedoch nicht, dass der Prozess des PFV als stark strukturiert oder gar standardisiert betrachtet werden kann. Tatsächlich ist er durch zahlreiche Freiheitsgrade bestimmt, die – dem jeweiligen Fall angepasst – die konkrete Durchführung beeinflussen. Weiterhin sind eine Vielzahl unterschiedlicher Personengruppen beteiligt, wodurch eine zusätzliche Komplexität des Verfahrens entsteht [Laub89, Sp. 1754–1757; Enge96].

PFV sind nicht als in sich abgeschlossene Verfahren zu betrachten. Dies geht zum einen auf die föderalistische Struktur Deutschlands und somit der Zersplitterung der Rechtsgebiete zurück und zum andern auf starke Überschneidungen zu Raumordnungsverfahren und Bauordnungen. Weiterhin definieren Vorschriften zur Bedarfsfeststellung oder der Festlegung der Planungsgebiete Schnittstellen zu dem vorgelagerten Prozess der Planeinreichung [Hufe86, S. 250; HoSB01, S. 89–95].

Die Zielsetzung der PFV besteht darin, durch einen sog. Planfeststellungsbeschluss die behördliche Feststellung eines Planes zu erreichen [Laub89, Sp. 1754–1755]. Betroffen sind Bauvorhaben, die durch spezialgesetzliche Rechtsvorschrift angeordnet und als übergeordnete raumbedeutsame Fachplanungen bezeichnet werden. Rechtlich richtet sich das PFV zunächst nach den jeweiligen Fachgesetzen, wie etwa dem Bundesfernstraßengesetz (FStrG). Die Vorschriften der Verwaltungsverfahrensgesetze gelten lediglich subsidiär, falls in den entsprechenden Fachgesetzen Lücken bestehen [HoSB01, S. 21].

Aus den PFV ergeben sich spezifische Rechtswirkungen, wie Konzentrations-, Genehmigungs- und Gestaltungswirkung. Die Konzentrationswirkung besagt, dass der Planfeststellungsbeschluss (PFB) alle nach anderen Rechtsvorschriften erlassenen, öffentlich-rechtlichen Genehmigungen, Verleihungen, Erlaubnisse und Zustimmungen ersetzt und somit keine weiteren behördlichen Entscheidungen benötigt werden. Durch die Genehmigungswirkung des PFB wird festgestellt, dass das geplante Vorhaben einschließlich aller Folgemaßnahmen und in Hinblick auf die von ihm berührten öffentlichen Belange zulässig ist. Die Gestaltungswirkung bezieht sich auf die öffentlich-rechtlichen Beziehungen zwischen Träger und Betroffenen. So berechtigt der PFB den Vorhabensträger zur Durchführung des festgestellten Plans, bindet ihn aber ebenso an dessen Inhalt sowie an alle damit verbundenen Auflagen. Der Planfeststellungsbeschluss wird durch einen formalen Verwaltungsakt definiert, der neben den Rechten und Pflichten der Träger des Vorhabens und der öffentlichen Rechtsträger auch die Rechte und Interessen der durch das Vorhaben Betroffenen enthält [Badu95, S. 512–515; HoSB01, S. 64–69].

3.2 Repräsentation des Planfeststellungsverfahrens durch EPK-Modelle

Aufgrund der verschiedenen, den jeweiligen Sachgebieten entsprechenden Fachgesetze für Planungsvorhaben, können bspw. straßenbaurechtliche, personenbeförderungsrechtliche, eisenbahnrechtliche oder wasserstraßenrechtliche Planfeststellung unterschieden werden. Trotz der Zugrundelegung der unterschiedlichen Fachgesetze weisen diese unterschiedlichen Vorschriften und Verfahren zahlreiche gemeinsame Merkmale auf [HoSB01, S. 20–38]. Nachfolgend werden exemplarisch Teile des PFV dargestellt, die anschließend in einem EPK-Modell dargestellt werden. Dabei stellt die Modellierung auf Basis der gesetzlichen Vorschriften nur einen deduktiven ersten Schritt zu einem Referenzmodell dar. In zukünftigen, induktiven Schritten ist die Verifikation und gegebenenfalls Anpassung des erstellten Referenzmodells aufgrund der in verschiedenen Verwaltungen erhobenen Modelle notwendig.

Die Planfeststellung beginnt mit der Einreichung des Plans bei der Anhörungsbehörde. Hier wird zunächst die Funktion „Plan öffentlich auslegen“ näher erläutert (vgl. Abbildung 1). Nachdem der Plan bei der Anhörungsbehörde eingereicht ist, prüft die Anhörungsbehörde zunächst, ob die betroffenen Personengruppen bereits bekannt sind, da in diesem Fall im Einvernehmen mit diesen auf eine öffentliche Auslegung des Plans verzichtet werden kann. In diesem Fall muss den Beteiligten dennoch Einsicht in die Unterlagen gewährt werden. Von der Option, auf die Auslegung zu verzichten, wird jedoch selten Gebrauch gemacht. Dennoch kann sich die Anhörungsbehörde für eine öffentliche Auslegung der Planungsunterlagen entscheiden [HoSB01, S. 52]. In diesem Fall veranlasst die Anhörungsbehörde innerhalb eines Monats nach Eingang der vollständigen Planungsunterlagen die Auslegung des Plans bei den betroffenen Gemeinden, indem sie ihnen die entsprechenden Unterlagen zusendet. Vor der eigentlichen Auslegung des Plans muss die Gemeinde diese ortsüblich bekannt geben. Die Bekanntmachung des Plans in der Öffentlichkeit soll bei den Betroffenen Interesse für den Plan wecken und dazu anhalten, sich mit dem Plan auseinander zusetzten. Die Art der ortsüblichen Bekanntmachung richtet sich maßgeblich nach dem Landes- bzw. Ortsrecht. Die Gemeinden stehen nun in der Pflicht, den Plan innerhalb von drei Wochen nach Erhalt für die Dauer eines Monats auszulegen. Ist den Betroffenen Einsicht gewährt, endet der Prozess „Plan öffentlich auslegen“ [HoSB01, S. 46–48].

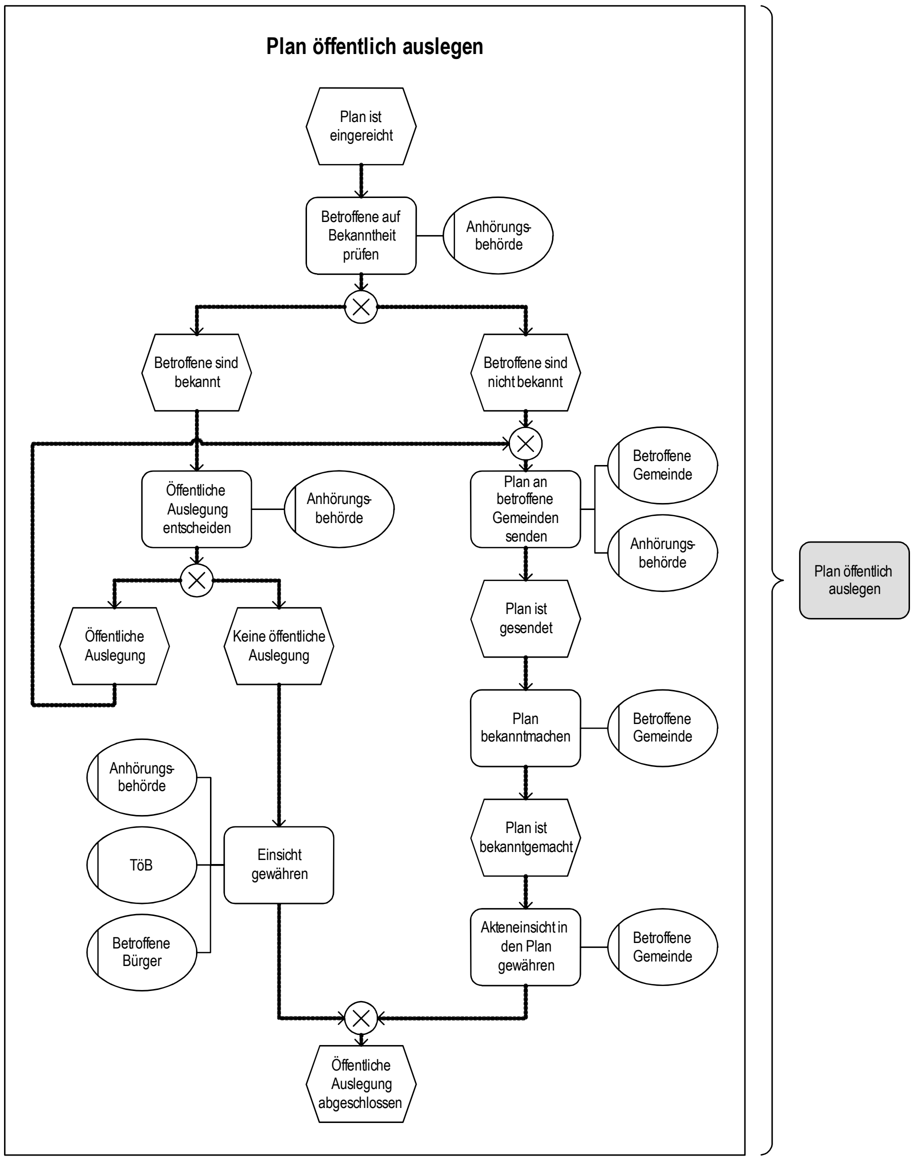
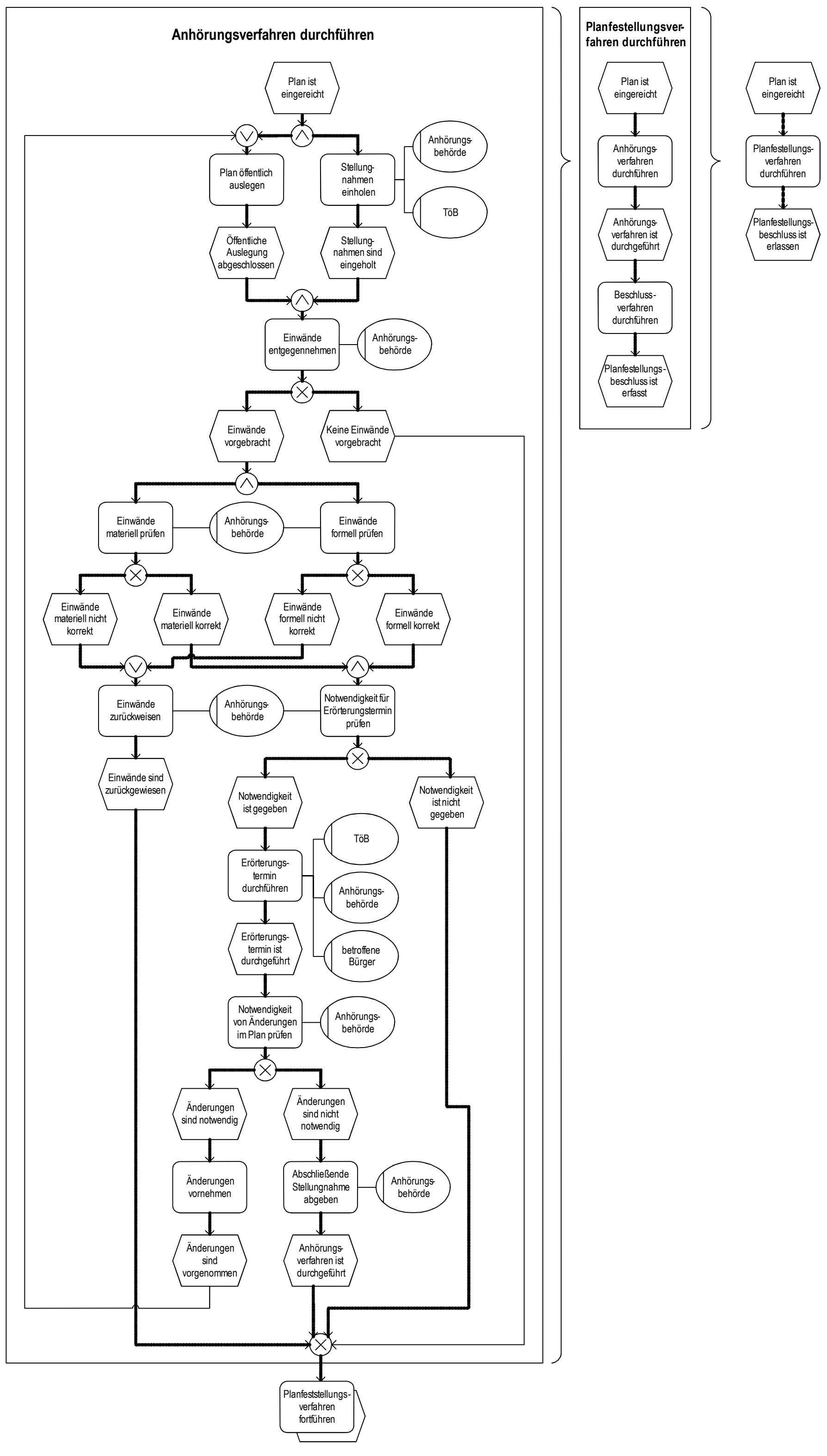


Abbildung 1: Öffentliche Auslegung

Der oben beschriebene Vorgang „Plan öffentlich auslegen“ ist Teil des Prozesses „Anhörungsverfahren durchführen“ (vgl. Abbildung 2). Der Prozess beginnt wiederum mit dem Startereignis „Plan ist eingereicht“. Die Anhörungsbehörde holt nun Stellungnahmen der Träger öffentlicher Belange (TöB) ein. Diese können etwa Industrie- und Handwerkskammern oder weitere Behörden wie Umweltamt, Gewerbeaufsichtsamt, Denkmalschutz- und Naturschutzbehörden sein. Parallel sorgt die Anhörungsbehörde für eine öffentliche Auslegung des Plans (vgl. Abbildung 1). Durch den Plan Betroffene können nun bis zwei Wochen nach Ablauf der Auslegungsfrist Einwände, Anregungen oder Änderungswünsche an den Plan erheben. Falls keine Einwände erhoben wurden, geht der Prozess direkt in den Teilprozess „Planfeststellungsverfahren durchführen“ über.



Anhörungsverfahren durchführen
Plan ist eingereicht
Plan öffentlich auslegen
Stellung-nahmen einholen
Anhörungs-behörde
TöB
Öffentliche Auslegung abgeschlossen
Stellung-nahmen sind eingeholt
Einwände entgegennehmen
Anhörungs-behörde
Einwände vorgebracht
Keine Einwände vorgebracht
Einwände materiell prüfen
Anhörungs-behörde
Einwände formell prüfen
Einwände materiell nicht korrekt
Einwände materiell korrekt
Einwände formell nicht korrekt
Einwände formell korrekt
Einwände zurückweisen
Anhörungs-behörde
Notwendigkeit für Erörterungstermin prüfen
Einwände sind zurückgewiesen
Notwendigkeit ist gegeben
Notwendigkeit ist nicht gegeben
TöB
Erörterungs-termin durchführen
Anhörungs-behörde
Erörterungs-termin ist durchgeführt
betroffene Bürger
Notwendigkeit von Änderungen im Plan prüfen
Anhörungs-behörde
Änderungen sind notwendig
Änderungen sind nicht notwendig
Änderungen vornehmen
Abschließende Stellungnahme abgeben
Anhörungs-behörde
Änderungen sind vorgenommen
Anhörungs-verfahren ist durchgeführt
Planfeststellungs-verfahren fortführen
Planfestellungsver-fahren durchführen
Plan ist eingereicht
Anhörungs-verfahren durchführen
Anhörungs-verfahren ist durchgeführt
Beschluss-verfahren durchführen
Planfestellungs-beschluss ist erfasst
Plan ist eingereicht
Planfestellungs-verfahren durchführen
Planfestellungs-beschluss ist erlassen

Abbildung 2: Anhörungsverfahren

Eingegangene Einwände werden zunächst einer Prüfung hinsichtlich ihrer formellen sowie materiellen Korrektheit unterzogen. Formelle Einwände sind bspw. Mängel in der schriftlichen Ausarbeitung des Einwandes. Die materielle Korrektheit betrifft inhaltliche Fragen des Plans wie bspw. die individuelle Betroffenheit der Einwanderhebenden. Falls die Einwände einen der aufgeführten Mängel aufweisen, hat die Anhörungsbehörde die Möglichkeit, die Einwände zurückzuweisen. Mit der Zurückweisung der Einwände geht der Prozess unmittelbar in die Schnittstelle „Planfeststellungsverfahren fortführen“ über. Sind die Einwände in allen Punkten korrekt, prüft die Anhörungsbehörde, ob die Notwendigkeit eines Erörterungstermins besteht. Der Erörterungstermin kann entfallen, falls sich alle Beteiligten gegen die Durchführung eines Erörterungstermins aussprechen und die beteiligten Behörden keine Bedenken äußern. Hier besteht wiederum die Verbindung zur Schnittstelle „Planfeststellungsverfahren fortführen“. An die Durchführung des Erörterungstermins schließt sich eine Prüfung des Plans auf Notwendigkeit von Änderungen an. Falls Änderungen in den Plan aufgenommen werden sollen, muss der Plan erneut öffentlich ausgelegt werden. Falls keine Änderungen vorgenommen werden, gibt die Anhörungsbehörde eine abschließende Stellungnahme ab, wodurch die Fortführung des PFV durch die Planfeststellungsbehörde eröffnet wird [HoSB01, S. 25–26].

Die Verbindung zwischen „Anhörungsverfahren durchführen“ und „Planfeststellungsverfahren fortführen“ wird in Abbildung 3 über die Prozessschnittstelle hergestellt.

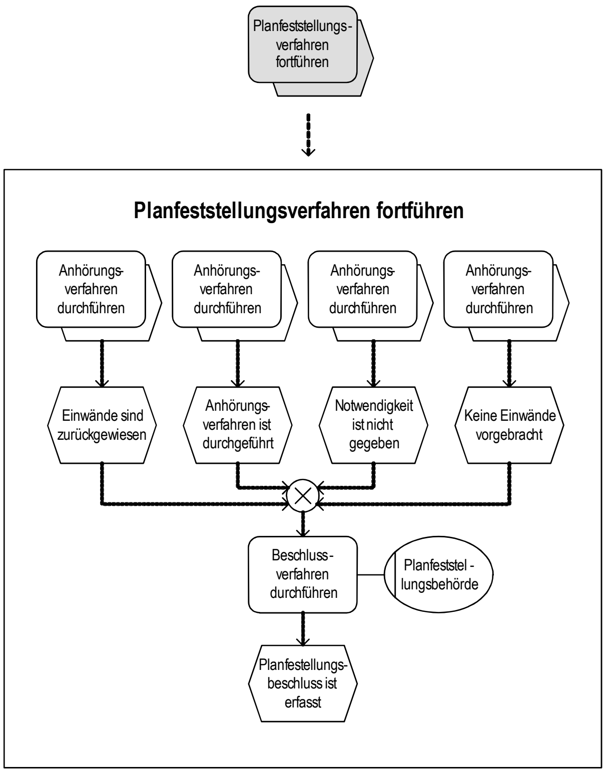

Planfeststellungs-verfahren fortführen
Planfeststellungsverfahren fortführen
Anhörungs-verfahren durchführen
Anhörungs-verfahren durchführen
Anhörungs-verfahren durchführen
Anhörungs-verfahren durchführen
Einwände sind zurückgewiesen
Anhörungs-verfahren ist durchgeführt
Notwendigkeit ist nicht gegeben
Keine Einwände vorgebracht
Beschluss-verfahren durchführen
Planfeststel-lungsbehörde
Planfestellungs-beschluss ist erfasst

Abbildung 3: Beschlussverfahren

Die Schnittstellen zwischen den beiden Prozessen sind als „Anhörungsverfahren durchführen“ bezeichnet, um den Übergang zu dem bereits dargestellten Prozessschritt zu verdeutlichen. Die Startereignisse stimmen mit den Endereignissen des abgeschlossenen

Prozesses „Anhörungsverfahren durchführen“ überein („Einwände sind zurückgewiesen“, „Anhörungsverfahren ist durchgeführt“, „Notwendigkeit ist nicht gegeben“, „Keine Einwände vorgebracht“). Es kann jeweils nur eines der genannten Ergebnisse Auslöser für die Weiterführung des Prozesses sein. In dem dargestellten Beispiel ist der Teilprozess in der Funktion „Beschlussverfahren durchführen“ enthalten. Das Beschlussverfahren endet mit dem Erlass des Planfeststellungsbeschlusses, wodurch der Prozess „Planfeststellungsverfahren fortführen“ abgeschlossen ist.

Analog zu der bereits beschriebenen Funktion „Plan öffentlich auslegen“ können weitere Funktionen wie etwa „Stellungnahmen einholen“, „Einwände materiell prüfen“, „Erörterungstermin durchführen“ usw. identifiziert werden. Diese Teilprozesse lassen sich auf einer höheren Ebene als „Anhörungsverfahren durchführen“ zusammenfassen. Die beiden Teilprozesse „Plan öffentlich auslegen“ und „Beschlussverfahren durchführen“ lassen sich nun zu dem übergeordneten Prozess „Planfeststellungsverfahren durchführen“ zusammenfassen.

3.3 Gestaltungspotenziale bei der Konstruktion von EPK-Referenzmodellen für Verwaltungsverfahren

Bei der Konstruktion der EPK-Referenzmodelle ergeben sich für die Modellierer Herausforderungen, die in den Charakteristika der öffentlichen Verwaltung begründet liegen. Diese müssen bei der Konstruktion der Referenzmodelle besondere Berücksichtigung finden, um eine adäquate Qualität der erstellten Modelle gewährleisten zu können.

Die juristische Verankerung der Prozesse wurde bereits in Kap. 2 aufgezeigt. Aufgrund der föderativen Struktur der Bundesrepublik Deutschland beruhen Regeln und gesetzliche Vorgaben für den Aufbau und Ablauf öffentlicher Verwaltungsorganisation meist nicht auf übereinstimmende Grundlagen. Vielmehr verteilen sich rechtliche Quellen auf eine Vielzahl von Gesetzen und Rechtsvorschriften [Kreu89, Sp. 1705]. So kann bspw. das Verwaltungsrecht unterschieden werden in das Allgemeine und das Besondere Verwaltungsrecht. Das Planungsrecht in dem oben dargestellten Anwendungsfall gehört zum Gebiet des Besonderen Verwaltungsrechts [Osse89, Sp. 1730]. Die Rechtsgrundlage bildet hier das Bundesverwaltungsverfahrensgesetz sowie die Verwaltungsverfahrensgesetze der Länder. [Laub89, Sp. 1753].

Der Modellierer muss dementsprechend zunächst eine Auswahl der relevanten rechtlichen Grundlagen treffen, anhand derer der Sachverhalt adäquat dargestellt wird. Es ist dabei aber zu berücksichtigen, dass prozessorientierte Beschreibungen in Gesetzen im Allgemeinen nicht üblich sind. Das Planfeststellungsverfahren wird zwar gemeinhin in zwei übergeordnete Verfahrensschritte (Anhörungsverfahren und Feststellungsverfahren) unterteilt. Sowohl die entsprechenden Gesetze als auch die Kommentare lassen auf diese Einschätzung schließen [HoSB01, S. 39 ff.]. Je tiefer der Modellierer nun aber auf den dargestellten Sachverhalt eingeht, desto mehr zeigt sich, dass für den Prozessablauf zusammenhängende Sachverhalte an unterschiedlichen Stellen der jeweiligen Quellen zu finden sind. So beschreibt § 73 VwVfG Abs. 6 das Erörterungsverfahren im Rahmen des

PFV. Für eine genauere Ausführung der Durchführungsmodalitäten wird jedoch auf § 68 VwVfG Abs. 1 Satz 3, Abs. 2 Nr. 1 und 4 und Abs. 3, § 68 VwVfG verwiesen.

Sind die für den ausgewählten Sachverhalt relevanten Gesetzestexte selektiert, benötigt ein Modellierer ohne juristische Vorkenntnisse zusätzliche Sekundärmaterialien, wie etwa Kommentare oder Beschreibung von Präzedenzfällen zum besseren Verständnis und zum Erfassen der möglichen Facetten des Prozesses. Das Verwaltensverfahrensgesetz (VwVfG) besagt etwa, dass jeder, dessen Belange durch das Vorhaben berührt wird, bis zwei Wochen nach Ablauf der Auslegungsfrist schriftlich oder zur Niederschrift bei der Anhörungsbehörde oder bei der Gemeinde Einwendungen gegen den Plan erheben kann [§ 73 Abs. 4 Satz 1 VwVfG]. Es stellt sich nun die Frage, was genau mit „dessen Belange berührt wird" gemeint ist. Erst durch das Hinzuziehen von Erläuterungen wird klar, dass eine „individuelle Betroffenheit" des Einwanderhebenden erkennbar sein muss, um den Einwand zur weiteren Prüfung zulassen zu können [HoSB01, S. 54].

Aufgrund der Diskrepanz zwischen Fach- und Modellierungssprachen kann auch das Studium relevanter Sekundärliteratur problematisch sein. Die besondere Terminologie des Planungsrechts beruht bspw. zum einen auf gesetzlichen Regelungen, zum anderen geht sie aber auf Entwicklungen in Rechtsprechung und Literatur zurück [HoSB01, S. 1]. So versteht ein Wirtschaftsinformatiker bspw. unter dem Begriff der Funktion die Verrichtung an einem Objekt, wodurch dieses erzeugt oder verändert wird [Sche97]. Die Sekundärliteratur zu PFV erfasst jedoch unter dem Kapitel „Funktionen der Planfeststellung", unter anderem die Kontroll- und Planungsfunktionen die erfüllt werden. Diese bezieht sich jedoch auf die benötigte staatliche Kontrolle, ohne deren Einfluss die zahlreichen Bauvorhaben nur schwer zu bewältigen sind [HoSB01, S. 9–13].

Die Ermessensspielräume bei der Bearbeitung erschweren ebenfalls eine Darstellung des Sachverhalts in einem Referenzmodell. So wird etwa in § 73 VwVfG Abs. 8 die Vorgehensweise im Falle von Planänderungen dargestellt. Die Sekundärliteratur erläutert die bestehenden gesetzlichen Bestimmungen dahingehend, dass bei geringfügigen Änderungen des Plans dessen erneute Auslegung nicht erforderlich ist [Badu95, S. 512].

Zusätzlich zu der Analyse der relevanten Verfahrensvorschriften und gesetzlichen Grundlagen muss eine Anreicherung der Modelle durch die relevanten Aspekte der Verwaltungspraxis erfolgen. Das Erfahrungswissen der Verwaltungsmitarbeiter kann somit in den Modellen dokumentiert und anderen Verwaltungen zur Verfügung gestellt werden. Die rechtlichen Vorgaben bilden allerdings eine unverzichtbare Grundlage und müssen als Ausgangspunkt in den Basismodellen dokumentiert werden.

Die dargestellten Besonderheiten der Verwaltungsverfahren erschweren die Konstruktion eines EPK-Referenzmodells und erfordern von dem Modellierer das Erfassen komplexer Zusammenhänge. Zur Erhöhung der Effektivität des Konstruktionsprozesses und zur Sicherstellung der Modellqualität wäre es vor dem Hintergrund der dargestellten Problemstellungen anzudenken, einen umfassenden Handlungsleitfaden unter Berücksichtigung der relevanten Domänenspezifika für die Erstellung von EPK-Referenzmodellen für die öffentliche Verwaltung zu entwickeln.

4 Fazit und Ausblick

Die Konstruktion eines Referenzprozessmodells für Planfeststellungsverfahren als Beispiel für Verfahren in der öffentlichen Verwaltung zeigt, dass textuelle oder natürlich sprachliche Prozessbeschreibungen, obwohl sie den Großteil des prozessrelevanten Wissens enthalten, nicht ohne erheblichen Aufwand in ein Prozessmodell umgesetzt werden können. Diese Schwierigkeiten bei der Modellkonstruktion resultieren zum einen aus den oben dargelegten Spezifika der Anwendungsdomäne, zum anderen liegen sie aber in der verwendeten Modellierungssprache begründet. Die EPK als Modellierungssprache enthält keine Handlungsanleitung, die ein operationalisiertes Vorgehen bei der Konstruktion der Modelle beschreibt. Eine solche Handlungsanleitung zur Modellkonstruktion wäre allerdings wünschenswert, da, wie das Beispiel der Planfeststellungsverfahren zeigt, die Konstruktion von EPK-Modellen kein rein intuitiver Akt ist.

Die strukturierte und systematische Konstruktion würde weiterhin zur Steigerung der Modellqualität beitragen. In diesem Zusammenhang wird unter „Modellqualität“ nicht die von bereits bestehenden Ansätzen, wie z. B. den GoM, betrachtete Qualität des Ergebnisses der Modellkonstruktion verstanden, sondern die Qualität des Vorgehens bei der Modellkonstruktion. Eine hohe Qualität des Vorgehens bei der Modellkonstruktion steigert die Nachvollziehbarkeit der Modelle und erleichtert so deren Änderbarkeit. Darüber hinaus wird die Entscheidung, ob ein von einem anderen Modellierer erstelltes Modell für einen bestimmten Zweck geeignet ist, da die Konstruktionsentscheidungen bei der Modellierung nachvollziehbarer werden. Die Autoren erhoffen sich daher eine fruchtbare Diskussion über die Erweiterung der EPK um operationalisierte Handlungsanleitungen, wobei ergänzend die Frage zu klären ist, ob Handlungsanleitungen allgemein, d. h. domänenunabhängig, dargestellt werden können oder ob sie eine domänenspezifische Ausrichtung erfahren müssen.

Literaturverzeichnis

[Badu95] Badura, P.: Das Verwaltungsverfahren. In: Erichsen, H.-U. (Hrsg.): *Allgemeines Verwaltungsrecht.* 10., neubearb. Aufl. Berlin [u. a.] : de Gruyter, 1995 (De-Gruyter-Lehrbuch), S. 415–519

[BeAN03b] Becker, J.; Algermissen, L.; Niehaves, B.: Prozessmodellierung als Grundlage des E-Government. Ein Vorgehensmodell zur prozessorientierten Organisationsgestaltung am Beispiel des kommunalen Baugenehmigungsverfahren. In: Uhr, W.; Esswein, W.; Schoop, E. (Hrsg.): *Wirtschaftsinformatik 2003 : Medien – Märkte, Mobilität.* Heidelberg : Physica-Verlag, 2003, S. 859–878

[BeAN03a] Becker, J.; Algermissen, L.; Niehaves, B.: Prozessmodellierung in eGovernment-Projekten mit der eEPK. In: Nüttgens, M.; Rump, F. J. (Hrsg.): *EPK 2003 : Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten ; Workshop der Gesellschaft für Informatik e.V. (GI) und Treffen ihres Arbeitskreises „Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)", Bamberg, 08. Oktober 2003, Proceedings.* Bonn : Gesellschaft für Informatik, 2003, S. 31–43. – URL http://epk.et-inf.fho-emden.de/epk2003/epk2003-proceedings.pdf [Zugriffsdatum 02.11.2003]

[BeAN04] Becker, J.; Algermissen, L.; Niehaves, B.: Vorgehensmodell zur Selektion von e-Government-Prozessen. In: *eGov-Präsenz* 4 (2004), Nr. 1, S. 4–8

[Bec+03] Becker, J. et al.: Konstruktion konfigurierbarer Referenzmodelle für die öffentliche Verwaltung. In: Dittrich, K. et al. (Hrsg.): *Informatik 2003 – Innovative Informatikanwendungen Band 1 : Beiträge der 33. Jahrestagung der Gesellschaft für Informatik e.V. (GI) ; 29.9.–2.10.2003 in Frankfurt am Main.* Bonn : Köllen, 2003 (GI-Edition – Lecture Notes in Informatics; P–34), S. 249–253

[Enge96] Engel, A.: Verwaltungsreorganisation mit Referenzmodellen. In: Scheer, A.-W. (Hrsg.): *Rechnungswesen und EDV : Kundenorientierung in Industrie, Dienstleistung und Verwaltung.* Heidelberg [u. a.] : Physica, 1996 (17. Saarbrücker Arbeitstagung 1996), S. 457–483

[Gi00] Gesellschaft für Informatik e.V. (GI); Informationstechnische Gesellschaft (ITG) (Hrsg.): *Electronic Government als Schlüssel zur Modernisierung von Staat und Verwaltung : Ein Memorandum des Fachausschusses Verwaltungsinformatik der Gesellschaft für Informatik e.V. und des Fachbereichs 1 der Informationstechnischen Gesellschaft im VDE.* Stuttgart : Alcatel SEL AG, 2000. – URL http://www.gi-ev.de/informatik/presse/presse_memorandum.pdf [Zugriffsdatum 23.12.2003]

[Hars94] Hars, A.: *Referenzdatenmodelle : Grundlagen effizienter Datenmodellierung.* Wiesbaden : Gabler, 1994 (Schriften zur EDV-orientierten Betriebswirtschaft). – Zugl.: Saarbrücken, Univ., Diss., 1993

[Heib04] Heib, R.: Effiziente Verwaltungsprozesse durch E-Government. In: Scheer, A.-W. et al. (Hrsg.): *Innovation durch Geschäftsprozessmanagement : Jahrbuch Business Process Excellence 2004/2005.* Berlin [u. a.] : Spinger, 2004, S. 385–398

[HoSB01] Hoppe, W.; Schlarmann, H.; Buchner, R.: *Rechtsschutz bei der Planung von Straßen und anderen Verkehrsanlagen.* 3., vollst. neu bearb. Aufl. München : Beck, 2001 (Schriftenreihe der Neuen juristischen Wochenschrift; 8)

[Hufe86] Hufen, F.: *Fehler im Verwaltungsverfahren.* Baden-Baden : Nomos, 1986

[KeNS92] Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)". In: Scheer, A.-W. (Hrsg.): *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, Nr. 89, Saarbrücken : Universität des Saarlandes, 1992. – URL http://www.iwi.uni-sb.de/Download/iwihefte/heft89.pdf [Zugriffsdatum 20.02.2003]

[KeTe96] Keller, G.; Teufel, T.: *SAP R/3 prozeßorientiert anwenden : iteratives Prozeß-Prototyping zur Bildung von Wertschöpfungsketten.* 1. Aufl. Bonn [u. a.] : Addison-Wesley, 1996 (Edition SAP)

[Krau83] Kraus, H.: Neue Informationstechnologien als Chance und Herausforderung für die öffentliche Verwaltung. In: Traunmüller, R. et al. (Hrsg.): *Neue Informationstechnologien und Verwaltung : Fachtagung 14.–16. September 1983.* Berlin [u. a.] : Springer, 1983 (Informatik Fachberichte), S. 15–27

[Kreu89] Kreuser, K.: Verwaltungsorganisation. In: Chmielewicz, K.; Eichhorn, P. (Hrsg.):

Handwörterbuch der Öffentlichen Betriebswirtschaft. Stuttgart : Poeschel, 1989 (Enzyklopädie der Betriebswirtschaftslehre; 11), S. 1697–1706

[Laub89] Laubinger, H.-W.: Verwaltungsverfahren. In: Chmielewicz, K.; Eichhorn, P. (Hrsg.): *Handwörterbuch der Öffentlichen Betriebswirtschaft.* Stuttgart : Poeschel, 1989 (Enzyklopädie der Betriebswirtschaftslehre; 11), S. 1753–1760

[Lenk99] Lenk, K.: Information und Verwaltung. In: Lenk, K.; Traunmüller, R. (Hrsg.): *Öffentliche Verwaltung und Informationstechnik : Perspektiven einer radikalen Neugestaltung der öffentlichen Verwaltung mit Informationstechnik.* Heidelberg : v. Decker, 1999 (Schriftenreihe zur Verwaltungsinformatik), S. 1–20

[Mauc99] Mauch, S.: *Qualitätsmanagement und lernende Organisation in der Landesverwaltung Baden-Württemberg.* Stuttgart : Stabsstelle für Verwaltungsreform im Innenministerium Baden-Württemberg, 1999 (StaV-Schriften). – URL http://www.verwaltungsreform-bw.de/cgi-bin/DxML?Template=./Templates/Start.tpl&DVA_Location=verwaltung [Zugriffsdatum 22.06.2004]

[Mayn68] Mayntz, R.: Max Webers Idealtypus der Bürokratie und die Organisationssoziologie. In: Mayntz, R. (Hrsg.): *Bürokratische Organisation.* Köln : Kiepenheuer & Witsch, 1968, S. 27–35

[NüRu02] Nüttgens, M.; Rump, F. J. (Hrsg.): *EPK 2002 : Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten ; Workshop der Gesellschaft für Informatik e.V. (GI) und Treffen ihres Arbeitskreises „Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)“, 21.–22. November in Tier, Proceedings.* Bonn : Gesellschaft für Informatik, 2002

[Nütt95] Nüttgens, M.: *Koordiniert-dezentrales Informationsmanagement : Rahmenkonzept – Koordinationsmodelle – Werkzeug-Shell.* Wiesbaden : Gabler, 1995 (Schriften zur EDV-orientierten Betriebswirtschaft). – Zugl.: Saarbrücken, Univ., Diss., 1995. – URL http://epk.et-inf.fho-emden.de/literatur/1995/Diss_MN.pdf [Zugriffsdatum 03.04.2001]

[Nütt97] Nüttgens, M.: Ereignisgesteuerte Prozeßkette (EPK) – Forschungsansätze in der wissenschaftlichen Literatur und Praxis. In: Desel, J.; Reichel, H. (Hrsg.): *Grundlagen der Parallelität : Workshop der GI-Fachgruppen 0.0.1 und 0.1.7 im Rahmen der Informatik '97.* Technische Universität Dresden, 1997 (Technische Berichte, Technische Universität Dresden, Fakultät Informatik; 97,13). – URL http://www.iwi.uni-sb.de/nuettgens/Veroef/Artikel/Gi-ws_97/Gi-ws_97.ps [Zugriffsdatum 02.11.2003]

[Osse89] Ossenbühl, F.: Verwaltungsrecht. In: Chmielewicz, K.; Eichhorn, P. (Hrsg.): *Handwörterbuch der Öffentlichen Betriebswirtschaft.* Stuttgart : Poeschel, 1989 (Enzyklopädie der Betriebswirtschaftslehre; 11), S. 1730–1738

[Reic87] Reichard, C.: *Betriebswirtschaftslehre der öffentlichen Verwaltungen.* 3. Aufl. Berlin : de Gruyter, 1987

[Rein86] Reinermann, H.: The Design of Information Systems for Local Administrations: From Bauhaus to Rathaus. In: *Computers, Environment and Urban Systems* 11 (1986), Nr. 1–2, S. 73–80. – URL http://www.elsevier.nl/inca/publications/store/3/0/4/

[Rein95] Reinermann, H.: Perspektiven einer Verwaltungsreform mittels Informationstechnik. In: Traunmüller, R. (Hrsg.): *Geschäftsprozesse in öffentlichen Verwaltungen : Neugestaltung mit Informationstechnik.* Heidelberg : R. v. Decker's, 1995, S. 125–139

[Ritt00] Rittgen, P.: Paving the Road to Business Process Automation. In: Hansen, H. R. (Hrsg.): *Proceedings of the 8th European Conference on Information Systems : Vienna University of Economics and Business Administration, Austria, 3rd–5th July 2000 ; (ECIS 2000).* Wien : Wirtschaftsuniv., 2000, S. 313–319

[Sche97] Scheer, A.-W.: *Wirtschaftsinformatik : Referenzmodelle für industrielle Geschäftsprozesse.* 7., durchges. Aufl. Berlin [u. a.] : Springer, 1997

[Sche02] Scheer, A.-W.: *ARIS – Vom Geschäftsprozess zum Anwendungssystem.* 4., durchges. Aufl. Berlin [u. a.] : Springer, 2002

[Schl00] Schlagheck, B.: *Objektorientierte Referenzmodelle für das Prozess- und Projektcontrolling : Grundlagen – Konstruktion – Anwendungsmöglichkeiten.* Wiesbaden : DUV, 2000 (Gabler Edition Wissenschaft: Informationsmanagement und Controlling). – Zugl.: Münster (Westfalen), Univ., Diss., 1999

[Schü98] Schütte, R.: *Grundsätze ordnungsmäßiger Referenzmodellierung : Konstruktion konfigurations- und anpassungsorientierter Modelle.* Wiesbaden : Gabler, 1998 (Neue betriebswirtschaftliche Forschung; 233). – Zugl.: Münster (Westfalen), U-niv., Diss., 1997

[ScNZ96] Scheer, A.-W.; Nüttgens, M.; Zimmermann, V.: Business Process Reengineering in der Verwaltung. In: Scheer, A.-W.; Friedrichs, J. (Hrsg.): *Innovative Verwaltungen 2000.* Wiesbaden : Gabler, 1996 (Schriften zur Unternehmensführung), S. 11–29

[ScPr03] Schedler, K.; Proeller, I.: *New Public Management.* 2. Aufl. Bern [u. a.] : Haupt UTB, 2003

[ScSc04] Schedler, K.; Schmidt, B.: Managing the E-Government Organization. In: *International Public Management Review* 5 (2004), Nr. 1, S. 1–20. – URL http://www.unisg.ch/org/idt/ipmr.nsf/ [Zugriffsdatum 12.06.2004]

[SeGü02] Seel, C.; Güngöz, Ö.: *E-Goverment: Strategien, Prozesse, Technologien, Studie und Marktübersicht (Oktober 2002).* Saarbrücken : IDS Scheer AG, 2002 (IDS Scheer Studien)

[Thie84] Thieme, W.: *Verwaltungslehre.* 4. Aufl. Köln [u. a.] : Heymanns Verlag, 1984

[vBro03] vom Brocke, J.: *Referenzmodellierung : Gestaltung und Verteilung von Konstruktionsprozessen.* Berlin : Logos, 2003 (Advances in information systems and management science; 4). – Zugl.: Münster (Westfalen), Univ., Diss., 2002. – URL http://www.wi.uni-muenster.de/aw/brocke/referenzmodellierung.pdf [Zugriffsdatum 30.05.2003]

[vLRe01] von Lucke, J.; Reinermann, H.: *Speyrer Definition von Electronic Government.* Speyer : Deutsche Hochschule für Verwaltungswissenschaften Speyer, 2001. – URL http://foev.dhv-speyer.de/ruvii/Sp-EGov.pdf [Zugriffsdatum 03.06.2004]

[WiTL01] Wimmer, M.; Traunmüller, R.; Lenk, K.: Prozesse der öffentlichen Verwaltung: Besonderheiten in der Gestaltung von e-Government. In: Horster (Hrsg.): *Elektronische Geschäftsprozesse: Grundlagen, Sicherheitsaspekte, Realisierungen, Anwendungen. Tagungsband zur gemeinsamen Arbeitskonferenz GI/VOI/BITKOM/OCG/TeleTrusT.* Höhenkirchen : it Verlag, 2001, S. 436–445

[WiTr02] Wimmer, M.; Traunmüller, R.: Geschäftsprozessmodellierung im E-Government: eine Zwischenbilanz. In: Schweighofer, E.; Menzel, T.; Kreutzbauer, G. (Hrsg.): *IT in Recht und Staat. Aktuelle Fragen der Rechtsinformatik 2002.* Wien : Verlag Österreich, 2002 (Schriftenreihe Rechtsinformatik), S. 19–27

EPML2SVG - Generating Websites from EPML Processes

Jan Mendling, Alberto Brabenetz, and Gustaf Neumann
Department of Information Systems and New Media
Vienna University of Economics and BA
`firstname.lastname@wu-wien.ac.at`

Abstract: This paper presents an approach to map EPC business process models available in EPML to Scalable Vector Graphics (SVG) and websites. This mapping has been implemented as an XSLT transformation program called EPML2SVG. Such a transformation may serve as a reference for visual tools that use EPML. Furthermore, business process models available in SVG can be used in web presentations and they can be viewed without buying a licence of a business process modelling tool. Moreover, EPML2SVG leverages EPML as an interchange format for EPCs. Websites generated by EPML2SVG contain SVG graphics for each EPC model and an HTML navigation structure based on links to each process models. We discuss design decisions of the program and illustrate the generated SVG by an example.

1 Introduction

Event-Driven Process Chains (EPC) are a wide-spread technique for modelling business processes. EPML (EPC Markup Language) is a tool-neutral interchange format for business process models represented as EPCs. It has been designed to serve as an import and export format as well as an intermediary format for transformations between heterogenous business process modelling tools [MN02, MN03, MN04]. It is related to other standardization efforts like OMG's XML Metadata Interchange (XMI) [Ob03], the Petri Net Markup Language (PNML) [BCvH 03], or WfMC's XML Process Definition Language (XPDL) [Wo02] that all aim to provide a tool-neutral interchange format.

In this paper we address the task of reusing business process models available in EPML and generating websites including Scalable Vector Graphics (SVG) [FJJ03] files from them. There are different motivations for this work. First, a transformation to SVG can serve as a reference model standardizing the visual representation of EPML documents (see [BCvH 03]). Second, SVG graphics of EPC business process models can be integrated into web-based trainings for e.g. SAP courses (see [As02]). Third, an SVG representation can be used within a website to communicate business process models to members of a company without buying a licence of a business process modelling tool like e.g. ARIS Toolset. Finally, the provision of tools is important to leverage EPML as an interchange format for EPCs.

EPML has been designed following the design principles of readability, extensibility, tool orientation, and syntactical correctness [MN03]. In the context of this work the EPML design principles of readability and tool orientation are especially important. First, readability provides for an easy development of programs that transform EPML code. Second, tool orientation in terms of position information of EPC symbols allows to build graphics from models available in EPML. This paper will present a transformation program that generates websites from EPML files. It is called EPML2SVG. This program is written in XSLT [Cl99], a scripting language specialized on generating new XML documents (i.e. HTML and SVG in this case) from other XML input files (i.e. EPML). Each EPC business process model will be used to build a separate SVG file. The rest of the paper is structured as follows. Section 2 will give an overview of SVG and its principles and advantages. In Section 3 we will present the generation of websites from EPML files. Section 4 will present related research. Finally, Section 5 will give some concluding remarks and an outlook on future research.

2 Scalable Vector Graphics

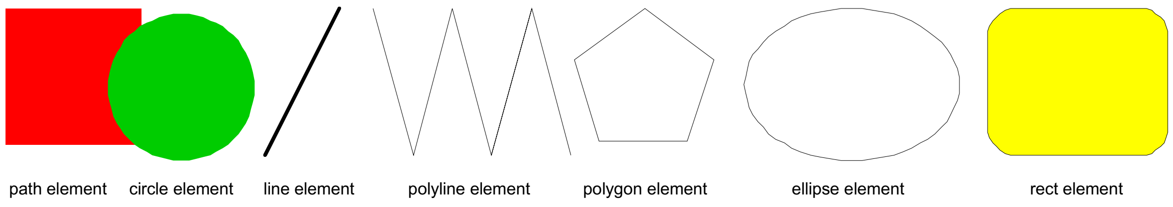
SVG is an XML-based format for two-dimensional vector graphics standardized by the World Wide Web Consortium [FJJ03]. It can contain graphic elements, images, and text. SVG defines a graphic as a set of geometric vector objects that can be grouped, styled, transformed, and composed (see [DHH02] for details). Vector representation provides for a loss-free scaling and storage efficiency. Furthermore, SVG offers dynamic effects like e.g. changing colors, moving text, or zooming. As SVG is based on XML it can be easily manipulated via a simple text editor. Moreover, SVG documents can be combined with other XML and programming technologies. On the one hand, SVG can be included in HTML documents [Pe02] and on the other hand it can include e.g. XLink elements [DMO01] to represent hyperlinks from graphic objects to web resources. Furthermore, text included in SVG graphics can be retrieved and indexed by search engines. A shortcoming is that SVG is not compressed like other image formats[1], because it is text-based. Yet, this is not a problem within the context of the work presented here.

SVG uses the so-called *painter's model* as its rendering concept (see [DHH02]). This term refers to the way an artist paints an oil painting. SVG paints the rendering elements in sequential order. It starts with the first element of the SVG document, let's say a red rectangle, and draws it on the area indicated by its coordinates. Second, a green circle is drawn whose area partially overlaps with that of the rectangle. According to the painter's model, the current element (the green circle) is drawn on top of the already existing drawing. This implies that the rectangle is partially obscured by the circle, unless semi-transparency has been defined for the circle. After that, the next element is drawn, and so on. When the circle has an edge, its area is drawn first, and the edge afterwards. This sequential drawing of elements on top of the current drawing constitutes the painter's model employed by SVG.

Each SVG document has an `svg` root element that includes a `width` and a `height` attribute defining the size of the graphic. The top left point represents the origin of SVG's

[1]For an overview of design criteria for multimedia interchange formats see [Ko92]

coordinate system. The `svg` element may nest multiple graphic elements including `rect`, `circle`, `ellipse`, `line`, `polygon`, `polyline`, `image`, `path`, `text`, and `use`. In the following we will sketch some of their characteristic. For a good introduction to SVG in general refer to [DHH02].



```
<svg width="900" height="300">
   <path d="M50,50 L140,50 L140,140, L50,140Z" fill="red"/>
   <circle cx="170" cy="100" r="50"  fill="green"/>
   <line x1="225" y1="150" x2="275" y2="50" stroke="black" stroke-width="5"  fill="none"/>
   <polyline points="300,50 325,150 350,50 375,150 400,50 425,150" stroke="black" fill="none"/>
   <polygon points=" 480,50 527,84 509,140 450,140 432,84" stroke="black" fill="none"/>
   <ellipse cx="620" cy="100" rx="72" ry="50" stroke="black" fill="none"/>
   <rect x="710" y="50" rx="20" ry="20" width="120" height="100" stroke="black" fill="yellow"/>
   <text x="50" y="180">path element</text>
   <text x="135" y="180">circle element</text>
   <text x="225" y="180">line element</text>
   <text x="320" y="180">polyline element</text>
   <text x="440" y="180">polygon element</text>
   <text x="580" y="180">ellipse element</text>
   <text x="735" y="180">rect element</text>
</svg>
```

Figure 1: Basic elements of SVG.

The most general graphic element of SVG is `path`. In Figure 1 the `path` element is used to draw a red rectangle. The `d` attribute defines the path by giving commands and references to positions. The `M` command moves the current position to the specified (x,y) coordinates. The `L` command draws a line from the current position to the specified position. `Z` can be used to close the path just like in the red rectangle example. See [DHH02] for further commands including cubic bezier curves. Basic shapes like `rect` or `ellipse` are simple shorthands for equivalent paths. Figure 1 illustrates how these shapes are defined: a `circle` by its center coordinates and radius, a `line` by its first and second point coordinates, `polyline` and `polygon` by a sequence of positions, an `ellipse` by its center coordinates and two radii, and a `rect` by its top left corner, width and height, and optionally the radii of elliptic arcs rounding the corners. Furthermore, text can be drawn by defining `text` elements (see Figure 1). The style of these shapes and text elements can be specified by various attributes. These include the definition of a fill color (`fill`), definition of a stroke color (`stroke`), or definition of the stroke width (`stroke-width`). For a complete list of these style attributes see the specification [FJJ03].

Furthermore, SVG permits images to be included using the `image` element. This is similar to the way as images are included in HTML. All elements can be grouped via a `g` element. The whole group can be translated, scaled, rotated, or skewed via a transform command. Finally, the `use` element draws a previously defined `symbol` on a specified positions on the SVG graphic. The SVG viewer internally replaces the `use` by a group element that contains the elements defined in the `symbol` element. The following section will discuss how SVG can be used to render EPC business process models defined in EPML.

3 From EPML to Web Presentations

This section introduces the EPML2SVG program implemented in XSLT. We will first give a short introduction to EPML. Then we discuss architectural alternatives to make EPML business process models available as websites. Afterwards, we present the mapping from EPML to websites implemented in the EPML2SVG program. Finally, we discuss the program. In case of potential confusion of names we use namespace prefixes to identify `svg` and `epml` elements and attributes.

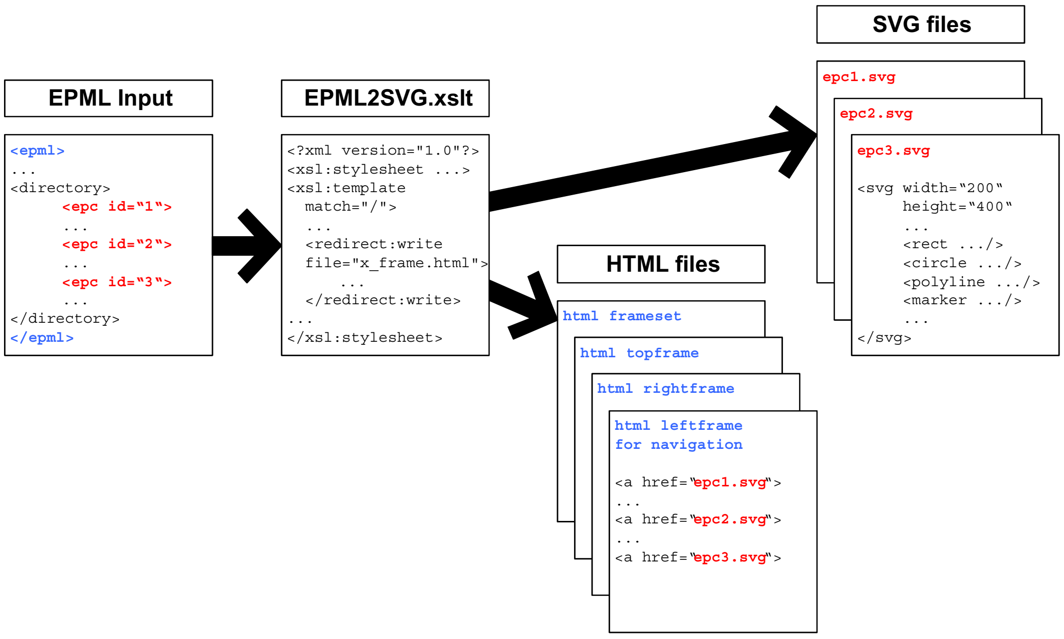
3.1 EPML in Brief

EPML is an XML-based tool-neutral interchange format for EPC business process models [MN02, MN03, MN04]. The `epml` element is the root of every EPML file. It contains among others a `directory` element that can nest further directories and `epc` models. Each of these models is identified by a unique `EpcId` and a `name` attribute. An `epc` element is a container for multiple control flow elements like `event`, `function`, `processInterface`, as well as `and`, `or`, and `xor` connectors, and multiple control flow `arc` elements. Each of these elements is identified by a unique `Id` attribute and a `name` element. The `function` and the `processInterface` element may include `ToProcess` elements. The latter has a `LinkToEpcId` attribute representing a logical pointer to a subprocess of a function or to a subsequent process of a process interface. Each `arc` element has a `flow` element whose `source` and `target` attributes represent the source and the target of the control flow arc. All EPC elements may have a `graphics` element. This element may contain `position`, `fill` (not applicable for arcs), `line`, and `font` visualization information. For control flow elements the `position` element specifies the `x` and the `y` position of the top left corner of a bounding box. Its size is indicated via the attributes `width` and `height`. Control flow arcs may have multiple `position` elements, each representing a point of a polyline. In the most simple case there are two position elements to represent the start point and the end point of the arc. For further details and further syntax elements of EPML we refer to [MN04].

3.2 Static versus Dynamic

Basically, there are two ways to generate websites from EPML files: static and dynamic. For a *static* website architecture all HTML and SVG files are generated in one go. Afterwards, these files can be published in a web directory. This approach has the advantage that it is easy to implement. Furthermore, requesting files from a static website provides a good performance. On the other hand, chances to the original EPML file will require a new run for generating the website. In a *dynamic* setting websites are generated from the original EPML file on request. That has the advantage that chances to the EPML file are immediately available. Yet, generating websites on the fly requires more computing time.

The EPML2SVG program provides for a generation of static websites. In particular, two different kinds of files are generated: HTML files for navigation through the EPC business process models, and SVG files for visualization of individual EPCs (see Figure 2). The EPML2SVG program arranges both kinds of files in an HTML frameset. Each EPML file maps to several HTML files. For one EPML file an HTML frameset is generated displaying a header frame at the top, a navigation frame at the left, and a welcome frame at the right. The navigation frame lists all EPC models of the EPML file and provides hyperlinks to corresponding SVG files to be displayed in the right frame. Furthermore, each `epml:epc` element maps to a separate SVG file whose name is built from its `epml:id` attribute. This naming convention is also used to generate the corresponding hyperlinks.



```
<epml>
...
<directory>
      <epc id="1">
      ...
      <epc id="2">
      ...
      <epc id="3">
      ...
</directory>
</epml>
```

```
<?xml version="1.0"?>
<xsl:stylesheet ...>
<xsl:template
  match="/">
  ...
  <redirect:write
  file="x_frame.html">
      ...
  </redirect:write>
...
</xsl:stylesheet>
```

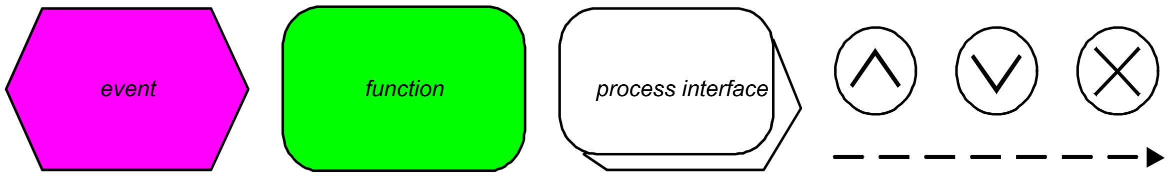
```
<svg width="200"
     height="400"
     ...
     <rect .../>
     <circle .../>
     <polyline .../>
     <marker .../>
     ...
</svg>
```

```
<a href="epc1.svg">
...
<a href="epc2.svg">
...
<a href="epc3.svg">
```

Figure 2: Files generated by the EPML2SVG program

3.3 Mapping from EPML EPCs to SVG

In general, there are two design choices to be taken concerning the representation of SVG elements generated from EPML files: path versus shape, and group versus symbol. The *first choice* is to be made between a generic representation using the `svg:path` element and a shorthand representation via basic shapes like e.g. `svg:rect`. We have decided to use basic shapes whenever possible because it avoids calculation in the transformation program. Consider the `epml:function` element. It is graphically described by its top left corner, and its height and width. This matches exactly the representation of a `svg:rect` element. For transforming it to a `svg:path` all corner coordinates would have to be calculated. The *second choice* is to be made between a group (`svg:g`) and a `svg:symbol` representation. In a group representation each element of an EPC is mapped to a group of SVG elements; e.g. an `epml:function` maps to a group containing a rectangle and a text label. The symbol representation is applicable for recurring complex sets of graphic elements. They can be defined in an `svg:symbol` which is referenced by a `svg:use` for each position where the symbol is to be displayed. As symbols in SVG cannot be parametrized for different text labels [DHH02], we chose for mapping each element of an EPC to a group of SVG graphic elements.



```
<svg height="80" width="540">
<g id="1" epc="event">
   <polyline transform="scale(0.4)" stroke-width="2" fill="#FF00FF" stroke="black"
      points="350,125 387.5,50 562.5,50 600,125 562.5,200 387.5,200 350,125"/>
   <text transform="scale(0.4)" fill="black" font-style="italic" font-size="12pt" y="125" x="400">event</text>
</g>
<g id="2" epc="function">
   <rect transform="scale(0.4)" fill="#00FF00" stroke="black" stroke-width="2"
      height="150" width="250" ry="20" rx="20" y="50" x="50"/>
   <text transform="scale(0.4)" fill="black" font-style="italic" font-size="12pt" y="125" x="100">function</text>
</g>
<g id="3" epc="processInterface">
   <rect transform="scale(0.4)" fill="none" stroke="black" stroke-width="2"
      height="135" width="222.5" ry="20" rx="20" y="50" x="650"/>
   <polyline transform="scale(0.4)" stroke-width="2" fill="none" stroke="black"
      points="872.5,80 900,143 862.5,200 700,200 675,185"/>
   <text transform="scale(0.4)" fill="black" font-style="italic" font-size="12pt" y="125" x="700">process interface</text>
</g>
<g id="4" epc="and">
   <circle transform="scale(0.4)" fill="none" stroke="black" stroke-width="2"
      height="100" width="100" r="50" cy="100" cx="1000"/>
   <polyline transform="scale(0.4) translate(0,7)" fill="none" stroke-width="5" stroke="black"
      points="970,110 1000,70 1030,110"/>
</g>
<g id="5" epc="or">
   <circle transform="scale(0.4)" fill="none" stroke="black" stroke-width="2"
      height="100" width="100" r="50" cy="100" cx="1150"/>
   <polyline transform="scale(0.4)" fill="none" stroke-width="5" stroke="black"
      points="1120,90 1150,130 1180,90"/>
</g>
<g id="6" epc="xor">
   <circle transform="scale(0.4)" fill="none" stroke="black" stroke-width="2"
      height="100" width="100" r="50" cy="100" cx="1300"/>
   <line transform="scale(0.4)" style="stroke-width:5;stroke:black" y2="130" x2="1330" y1="70" x1="1270"/>
   <line transform="scale(0.4)" style="stroke-width:5;stroke:black" y2="70" x2="1330" y1="130" x1="1270"/>
</g><g id="7" epc="arc">
   <polyline points="950,175 1350,175 " stroke="black" stroke-dasharray="0.25% 0.25% 0.25%"
      fill="none" transform="scale(0.4)" style="marker-end:url(#arrow)"/>
</g>
</svg>
```

Figure 3: EPC elements and SVG code generated by EPML2SVG.

Figure 3 displays EPC icons and their SVG representation generated by EPML2SVG. For each `function`, `event`, `processInterface`, `and`, `or`, `xor`, and `arc` element a `svg:g` group is generated containing icon-specific shape elements. The size and the position of the shape group is determined by the `epml:graphics` subelements of the EPC elements. In order to easily relate SVG shapes to their EPML source code, the EPML2SVG program writes the `epml:id` of the source element to the respective `svg:g` element and attaches an `svg:epc` attribute[2] stating its EPC element type. Accordingly, a `function` with `epml:id="2"` maps to a group element with a `svg:id="2"` and a `svg:epc="function"` attribute (see Figure 3).

[2]Note that this `epc` attribute is syntactically written to the `svg` namespace. Yet, of course, the SVG specification does not define an `epc` attribute for groups. The SVG viewer will ignore it. Accordingly, it is actually a comment.

In the following, we give only some short explanations on the mapping. For further details we refer to the EPML2SVG program[3]. In EPML functions and process interfaces can contain `toProcess` subelements whose `linkToEpcId` attribute represents a reference to another process. These EPML elements are mapped to XLink hyperlinks in the SVG graphic. Such hyperlink allow to open the referenced process graphic via a simple click. Furthermore, `svg:polyline` elements generated from arcs reference a `svg:marker` element. It permits to draw arrow heads for a line by simply referring to a path defined by the marker. For details see the specification [FJJ03]. The rest of the transformation are relatively straight forwards mappings to SVG shapes.

Figure 4 gives a screenshot of an HTML frameset generated by the EPML2SVG program. There are three EPC process models included in the navigation list at the left-hand side. Each contains a hyperlink to an SVG file of the corresponding EPC business process model. The right frame displays the SVG file of such an EPC model.

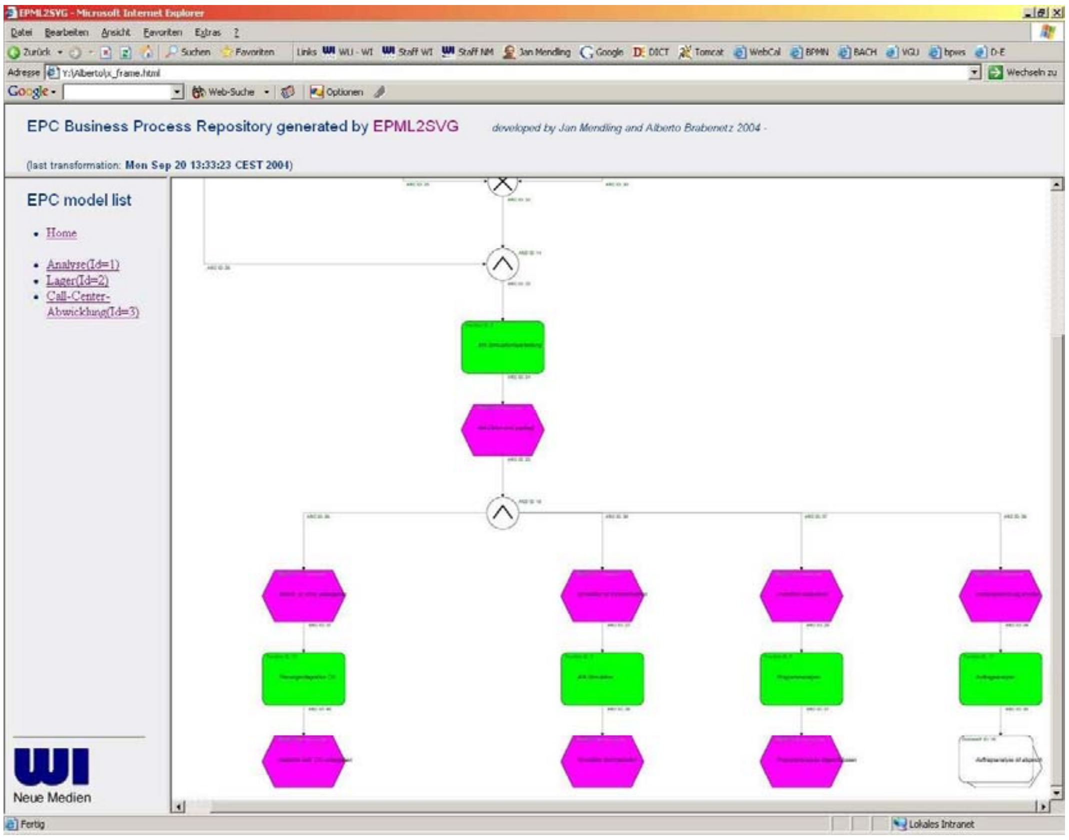

EPML2SVG - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Adresse Y:\Alberto\x_frame.html
EPC Business Process Repository generated by EPML2SVG
developed by Jan Mendling and Alberto Brabenetz 2004
(last transformation: Mon Sep 20 13:33:23 CEST 2004)
EPC model list
Home
Analyse(Id=1)
Lager(Id=2)
Call-Center-Abwicklung(Id=3)
Neue Medien
Fertig
Lokales Intranet

Figure 4: Screenshot of a HTML Frameset generated by the EPML2SVG program.

[3]Information on how to obtain the EPML2SVG program is available on the EPML website (http://wi.wu-wien.ac.at/˜mendling/EPML).

3.4 Discussion

Although most of the mappings can easily be implemented using XSLT, there are some challenges regarding texts. SVG does not understand newline characters in text elements. This means that each time there is a newline character in the text label of e.g. an `epml:function` a separate text span at a position below the first part of the string has to be added. In XSLT this has to be programmed by the help of a recursive template that processes the text label as long as there are no more newline characters[4]. If SVG were able to interpret newline characters, the transformation would be much more straight forward.

Another challenge is the automatic layering of process graphs, in particular EPC business process models. The EPML2SVG program is not able to calculate positions when there are no `epml:graphics` element included. Layout and drawing of graphs is a non-trivial problem and subject of a whole research community (see e.g. [BETT99]). The automatic calculation of positions is among others needed when an EPML file has been generated from a language like BPEL4WS [ACD 03] that does not include any graphical information. We expect to address this challenge in the future.

4 Related Research

There have been some academic research projects dealing with transformation of XML business process model data into graphics. In the context of Petri Net Markup Language (PNML) [BCvH 03] a mapping to SVG has been refined. This mapping is implemented in the PNML2SVG transformation script available at [St03a]. This mapping has been defined to provide for a precise description of the graphical presentation of a PNML document. PNML compliant visual tools can check their conformance using this reference model of PNML visualization. A project at TU Chemnitz uses the ARIS Markup Language (AML), the XML interchange format of ARIS Toolset, to generate SVG graphics of business process models [As02]. The purpose of this project is to make EPC business process models available in web-based SAP trainings. In contrast to EPML2SVG it transforms only one EPC model to one SVG file. Beyond that, EPML2SVG generates a whole HTML navigation frameset. The AML Interpreter project at Rostock University also takes AML as an input format [St03b]. The AML Interpreter displays EPC business process models available in AML and offers a transformation to standard graphic formats GIF and JPEG. The motivation of this project is to allow visualization of ARIS process models without having to install ARIS Toolset. This may safe licence cost. There are usually many employees in a company who only sometimes need to view process models. For them a simple viewer like AML Interpreter would be sufficient and no ARIS Toolset installation required. Yet, GIF and JPEG are not vector graphics, but image formats. In contrast, the SVG files generated by EPML2SVG provide for a loss-free scaling of the EPC business process models.

[4]For details on the recursive template solution see the EPML2SVG program available from the EPML website (http://wi.wu-wien.ac.at/~mendling/EPML).

5 Conclusion and Future Work

In this paper we have presented the EPML2SVG transformation program written in XSLT that generates websites including SVG graphic files from EPML documents. This transformation can be used as a reference model when implementing graphical EPC tools. Furthermore, the generated graphics can be used in web training courses on business process-related topics. Moreover, they can be used to publish process models on the intranet of a company without having to buy licences for expensive business process modelling tools. Finally, EPML2SVG leverages EPML as an interchange format for EPCs.

Although the exchange of graphical model information has been recognized as an important issue of interchange model data (see e.g. [MN04, BCvH 03, Ob03]) there is currently no consensus on how representing graphic information in such interchange formats. It will be an interesting topic of future research to identify the pros and cons of directly including SVG code in interchange formats. Furthermore, automatic calculation of positions for EPC business process model elements is a non-trivial task. We expect to address this problem in the context of EPML in the future.

Acknowledgement. The authors would like to thank the anonymous reviewers for there comments on an earlier version of this paper, which helped to improve the presentation of the ideas.

References

[ACD 03] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., and Weerawarana, S.: Business Process Execution Language for Web Services, Version 1.1. Specification. BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems. 2003.

[As02] Assmann, M.: *Generierung webfähiger EPK-Grafiken mittels ARIS, XML und SVG*. TU Chemnitz. http://www-user.tu-chemnitz.de/˜maas/projectpage/index2.html. 04.12.2002.

[BCvH 03] Billington, J., Christensen, S., van Hee, K. E., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., and Weber, M.: The Petri Net Markup Language: Concepts, Technology, and Tools. In: W. M. P. van der Aalst and E. Best (eds.), *Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN 2003, Eindhoven, The Netherlands*. volume 2679 of *Lecture Notes in Computer Science*. pages 483–505. 2003.

[BETT99] Battista, G. D., Eades, P., Tamassia, R., and Tollis, I. G.: *Graph Drawing - Algorithms for the Visualization of Graphs*. Prentice Hall. 1999.

[Cl99] Clark, J.: XSL Transformations (XSLT) Version 1.0. W3C Recommendation 16 November. World Wide Web Consortium. 1999.

[DHH02] Duce, D., Herman, I., and Hopgood, F.: SVG Tutorial. In: *Proceedings of the WWW2002 Conference, Hawaii, USA*. http://www.w3.org/2002/Talks/www2002-svgtut-ih/hwtut.pdf. May 2002.

[DMO01] DeRose, S., Maler, E., and Orchard, D.: XML Linking Language (XLink) Version 1.0. W3C Recommendation 27 June 2001. World Wide Web Consortium. 2001.

[FJJ03] Ferraiolo, J., Jun, F., and Jackson, D.: Scalable Vector Graphics (SVG) 1.1. W3C Recommendation 14 January 2003. World Wide Web Consortium. 2003.

[Ko92] Koegel, J. F.: On the Design of Multimedia Interchange Formats. In: *Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*. pages 262–271. 1992.

[MN02] Mendling, J. and Nüttgens, M.: Event-Driven-Process-Chain-Markup-Language (EPML): Anforderungen zur Definition eines XML-Schemas für Ereignisgesteuerte Prozessketten (EPK). In: M. Nüttgens and F. J. Rump (eds.), *Proc. of the 1st GI-Workshop on Business Process Management with Event-Driven Process Chains (EPK 2002), Trier, Germany*. pages 87–93. 2002.

[MN03] Mendling, J. and Nüttgens, M.: XML-basierte Geschäftsprozessmodellierung. In: W. Uhr, W. Esswein and E. Schoop (eds.), *Proc. of Wirtschaftsinformatik 2003 / Band II, Dresden, Germany*. pages 161 –180. 2003.

[MN04] Mendling, J. and Nüttgens, M.: Exchanging EPC Business Process Models with EPML. In: J. Mendling and M. Nüttgens (eds.), *Proc. of the 1st GI-Workshop XML4BPM - XML Interchange Formats for Business Process Management, Marburg, Germany, March, 2004*. pages 61–79. 2004.

[Ob03] Object Management Group: XML Metadata Interchange (XMI). Specification, Version 2.0. Object Management Group. May 2003.

[Pe02] Pemberton et al., S.: XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). W3C Recommendation 26 January 2000, revised 1 August 2002. World Wide Web Consortium. 2002.

[St03a] Stehno, C.: *PNML2SVG*. Universität Oldenburg. http://parsys.informatik.uni-oldenburg.de/˜pep/pnml. 28.02.2003.

[St03b] Stoy, M.: *AML Interpreter*. Universität Rostock. http://www.informatik.uni-rostock.de/˜masto/aris/. 01.03.2003.

[Wo02] Workflow Management Coalition: Workflow Process Definition Interface – XML Process Definition Language. Document Number WFMC-TC-1025, October 25, 2002, Version 1.0. Workflow Management Coalition. 2002.

ARIS meets RUP
The ARIS Unified Information System Development Process

Martin Plümicke
University of Cooperative Education Stuttgart
Department of Information Technology
Florianstraße 15
D–72160 Horb
tel. +49-7451-521142
fax. +49-7451-521190
m.pluemicke@ba-horb.de

Abstract: Starting from the Rational Unified Process RUP and the ARIS life cycle model the ARIS Unified Information System Development Process AUP is designed. Similar as the RUP is developed from the classic waterfall model as an iterative and incremental process to build software, the AUP is developed from the ARIS life cycle model as an iterative and incremental process to build business information systems. As software development is a part of the information system development the RUP is integrated in the AUP. The modeling language for the AUP is a combination of UML and diagrams from the ARIS framework concept.

1 Introduction

In last years great efforts are done to improve software development processes [Ba00]. Starting from the traditional *waterfall model* many other models, as *V–modell*, *prototyping–model*, and *The Rational Unified Process*, are developed. These models consider all almost only the software development.

In the same time the application of computer science in the business were extended from pure software solutions to large information systems, which implement complete business processes. For this beside software implementation, databases, networks, and its integration must be considered.

Software-engineering is presently almost done object–oriented. The *Unified Modeling Language* (UML) [FS00] defines many different model types to describe a software product from different views. UML itself is no software engineering process. For UML a software engineering process is developed by Rational Software: *The Rational Unified Process* (RUP) [JBR98, Kr03, RUP]. RUP differs in two points essentially from traditional software engineering processes like the waterfall model. On the one hand RUP has no static phases as analysis, design, implementation, and test. There are different workflows, which are applied iteratively. The main advatages of iterative software development

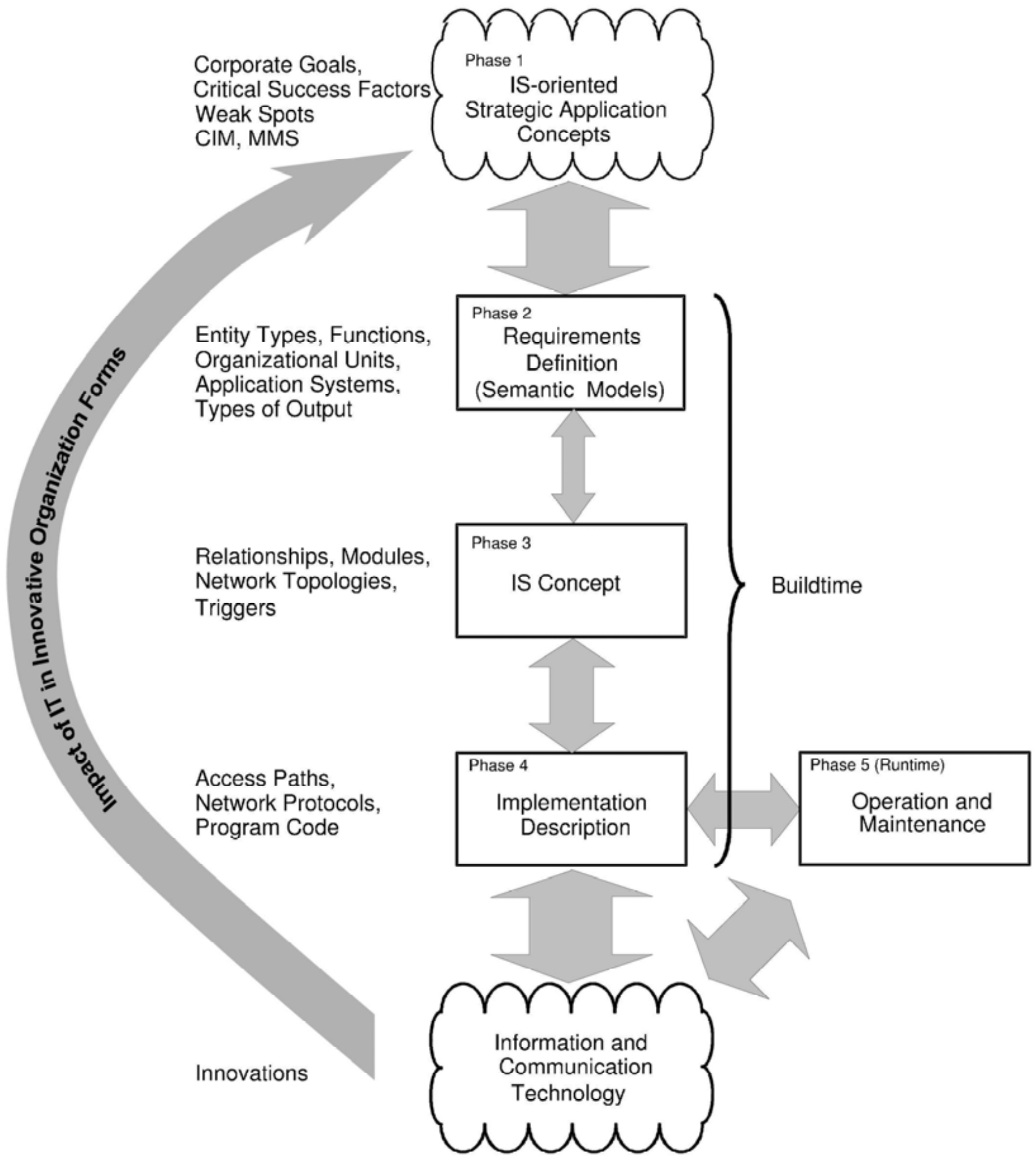

Corporate Goals,
Critical Success Factors
Weak Spots
CIM, MMS
Phase 1
IS-oriented Strategic Application Concepts
Impact of IT in Innovative Organization Forms
Entity Types, Functions, Organizational Units, Application Systems, Types of Output
Phase 2
Requirements Definition (Semantic Models)
Relationships, Modules, Network Topologies, Triggers
Phase 3
IS Concept
Buildtime
Access Paths, Network Protocols, Program Code
Phase 4
Implementation Description
Phase 5 (Runtime)
Operation and Maintenance
Innovations
Information and Communication Technology

Figure 1: ARIS life cycle model

is that aberrations are detected earlier, which can reduce costs enourmously. On the other hand RUP is a configurable process, which means that RUP can be varied to accommodate different situations.

The *ARchitecture of integrated Informations Systems* (ARIS) [Sc02, Sc01] is a framework concept to describe a complete business information system. ARIS is extended by a development process, the ARIS life cycle model, which is similar to traditional software engineering processes.

The content of this paper is the advancement of the ARIS life cycle model to a modern iterative process, which is enhanced from the RUP.

In this paper we will start in the next section by describing RUP and ARIS. In the third section we consider some related work, which deals with the integration of ARIS and UML. Then, in the fourth, the main section, we define the ARIS Unified Information System Development Process (short: ARIS Unified Process, abbreviated as AUP). Finally we close with a conclusion and an outlook.

2 Overview over ARIS and RUP

2.1 ARIS

The ARIS framework concept considers business information systems by five different views.

Function view: In the function view all functions are modeled. Functions are defined as transformations from input products to output products.

Organization view: In the organization view the structure of the company is modeled.

Data view: In the data view the data are modeled which are input or output products, which are environment data, and which are events or messages in the control view.

Output view: In the output view products are modeled, which are either produced or bought for ennobling.

Control view: In the control view on the one hand the connections between the different views are modeled. This is the static part of the control view. On the other hand the dynamic part of the business process is modeled. This is done by the Event–Driven Process Chains (EPCs) [KNS92].

For the building of a business information system each of these five views are considered by the ARIS life cycle model (figure 1)[1].

[1]The figure is taken from `www.iwi.uni-sb.de` as a translation of a figure in [Sc02].

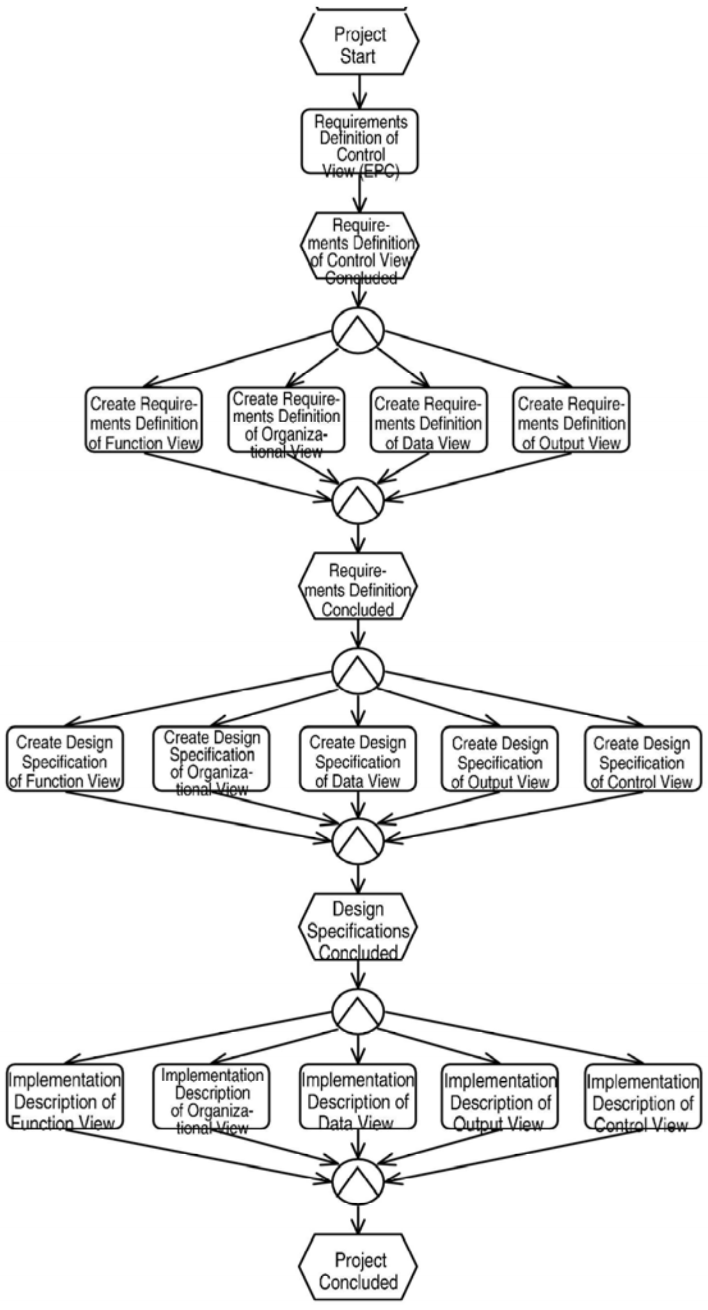

Project Start
Requirements Definition of Control View (EPC)
Requirements Definition of Control View Concluded
Create Requirements Definition of Function View
Create Requirements Definition of Organizational View
Create Requirements Definition of Data View
Create Requirements Definition of Output View
Requirements Definition Concluded
Create Design Specification of Function View
Create Design Specification of Organizational View
Create Design Specification of Data View
Create Design Specification of Output View
Create Design Specification of Control View
Design Specifications Concluded
Implementation Description of Function View
Implementation Description of Organizational View
Implementation Description of Data View
Implementation Description of Output View
Implementation Description of Control View
Project Concluded

Figure 2: Rough ARIS process

The five phases of the ARIS life cycle model are the basis for the workflows of the ARIS unified process in section 4. The rough ARIS process for modeling a business information system is shown in figure 2.[2] This life cycle model is very similar to the waterfall model. The main problem of the waterfall model is, that the risks of the system development are consequently shifted in later phases. But fixing of detected defects becomes during the software process more expensive. This means the risks should be managed as early as possible.

The ARIS framework concept and the life cycle model is summarized in the ARIS house (figure 3)[3]. In the ARIS house only the second, third, and fourth step of the ARIS life cycle

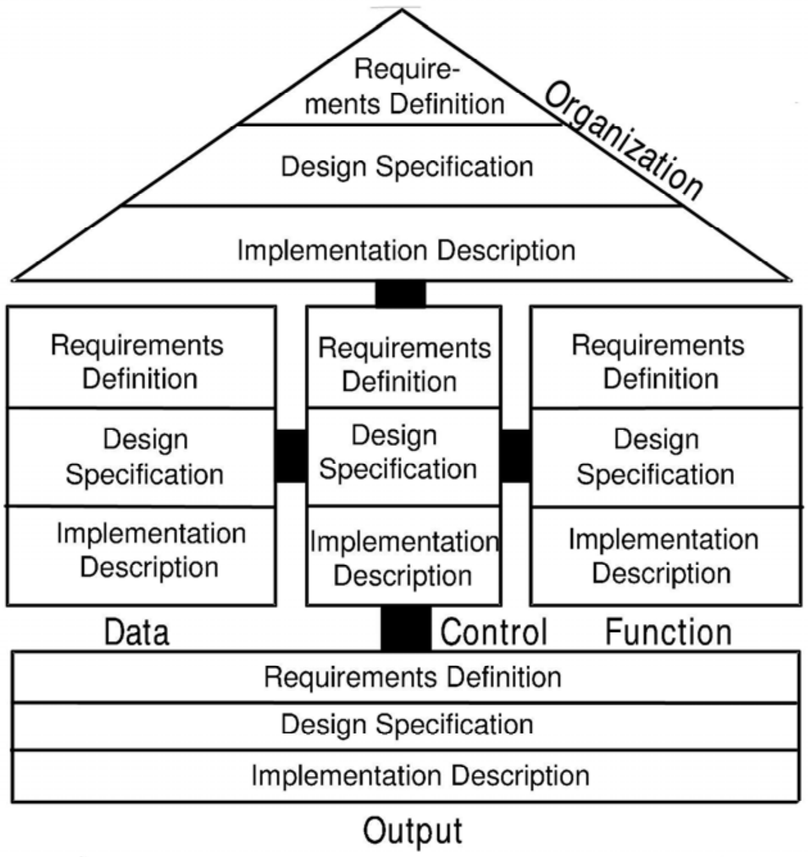


Figure 3: The ARIS house

are considered. The further steps are considered together for all views of the information system.

For each phase of each view there is at least one diagram type to model the view of the phase. The control view is divided in a static and a dynamic part. In the static part for each connection of different views there is a diagram type to model it. The dynamic part is modeled by the EPCs. We do not present these diagram types at this point of the paper as the diagram types are given during the description of the ARIS Unified Process in

[2]The figure is taken from `www.iwi.uni-sb.de` as a translation of a figure in [Sc02].
[3]The figure is taken from `www.iwi.uni-sb.de` as a translation of a figure in [Sc02].

section 4.

In the framework of this paper we do not consider the *output view*. The reason is, that the development of a business information system either only supports the product development or the products are data. In the first case the information system supports the production in some production steps. But these steps are defined in the function view. In the second case the products are considered in the data view.

2.2 RUP

The main characteristics of the Rational Unified Process RUP[4] is that it is iterative and incremental. This means that large software development projects are broken down into a number of smaller *mini projects*, which are easier to manage. The key point is that each iteration contains all elements of a standard software development project: *planing*, *analysis and design*, *construction*, *integration and test*.

This property leads to a two dimensional description of the process (fig. 4)[5]:

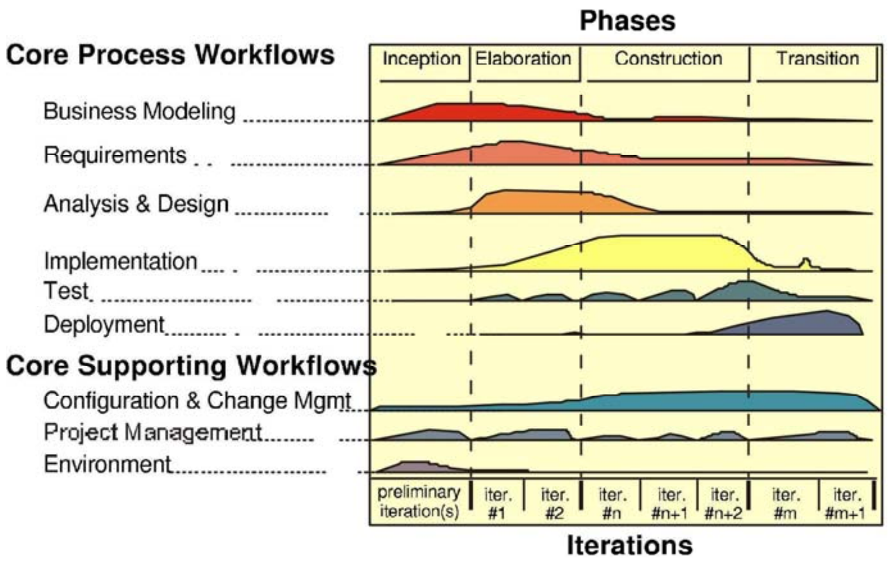


Figure 4: RUP along two dimensions

- The horizontal axis represents time. It is the dynamic aspect of the process. The process is divided in phases and iterations.
- The vertical axis represents the static aspect of the process: the workflows, the activities, the workers, and the artefacts.

[4]The content of this section is taken from [RUP, JBR98, Kr03]

[5]The figure is taken from [RUP].

The diagram itself shows how much work is done of which workflow in the different phases.

2.2.1 The dynamic structure of the RUP– the phases

The software development project is divided in four phases. Each phase is again divided in severals iterations. Each iteration is closed by a milestone and each phase is closed by a major milestone.

In the description of the different phases we will refer to the workflows, the workers, and the artefacts, as they are orthogonal to the phases. These aspects are considered in the following sections.

Inception Phase: The purpose of the inception phase is to get a base for the project. This involves the following aspects:

- establishing feasibility (e.g. prototyping to validate the business requirements)
- creating business case to deliver quantifiable business benefit
- capturing essential requirements to help scope the system
- identifying critical risks

The primary emphasis is on the requirements and the analysis workflow. However some design and implementation as prototypes also might be done. The test workflow is not applied, as the only software artefacts are prototypes, which are later thrown away.

The primary workers of the inception phase are the project manager and the system architect.

The milestone is the *Life Cycle Objective*, which includes the following conditions, such the milestone is reached: stakeholder have agreed the project objectives, system scope have been defined (10% – 20% of the use–cases), key requirements have been captured, costs and risks have been assessed, confirmation of feasibility, and the architecture is outlined.

Elaboration Phase: The purposes of the elaboration phase are summarized as follows:

- create an executable architectural baseline
- refine the risk assessment
- capture use–cases to 80%
- create a detailed construction plan including resources and costs

The focus of the elaboration phase is the creation of an executable architectural baseline. While in the design workflow a stable architecture is created, in the implementation workflow the architecture baseline is constructed. This is no prototype, rather a first cut out of the design system.

The milestone is the *Life Cycle Architecture*, which includes the vision document, an executable architectural baseline demonstrates that the important risks are identified and resolved, a realistic project plan with statements of time, money, and resources, and an agreement with the stakeholders to continue the project.

Construction Phase: The purposes of the construction phase are given as:

- complete the requirements
- finish the analysis model
- finish the design model
- evolve the architectural baseline from the elaboration phase into the final system

The focus in this phase is on the implementation workflow. Testing becomes also more important.

The milestone is the *Initial Operational Capability*, which includes the software product, the UML model, a test suite, and a user manual. The actual expenditures respectively the planned expenditures are acceptable. The stakeholders have agreed to the transition to their environment.

Transition Phase: The transition phase starts when beta testing is completed. The purposes can be summarized as follows:

- correct defects
- prepare the user site for the new software
- prepare documentation
- provide user consultancy

The emphasis is on the implementation and the test workflow. Hopefully, by this point, there should be very little work being done in the requirements and the analysis workflow.

The milestone is the *Product Release*, which includes the software product, the actively using of the product by the users, the user support plans, and the user manuals.

Each phase can be broken down into iterations. An *iteration* is a complete development loop resulting in a release of an executable product, which grows incrementally to become the final system. The benefit of the iterative approach is, that risks are migrated earlier, changes are more manageable, there is a higher level of reuse, and the quality is better overall.

2.2.2 The static structure of the RUP

The static structure is represented by using four modeling elements: *the workers*, who is doing something, *the activities*, how it is done, *the artefacts*, what is done, and *the workflows*, when it is done.

The workers

A *worker* defines the behavior and responsibilities of an individual, or a group of individuals working together as a team. In the RUP the worker is more a role defining how the individuals should carry out the works than the individuals itselves.

The activities

The activities are asked to perform by a specific worker. An activity is considered as creation or updating an artefact. For example *plan an iteration* is assigned to the project manager or *find use–cases and actors* is assigned to the system analyst.

The artefacts

An artefact is a piece of information that is produced, modified, or used in process. Artefacts are used by a worker as input for an activity, and as results of the activity. Artefacts are for example models (use–case model, analysis model), documents, source code, and executables.

The workflows

In the RUP the workflows (fig. 4) are divided in two groups: the core process workflows and the core supporting workflows. In this section of the paper we will give only a short overview of the workflows, as in section 4 the workflows of the AUP, which are defined as an extension of the RUP workflows, are considered more detailed.

Business Modeling: One of the major problems is that the software engineering and the business engineering community do not communicate properly with each other. The purpose of the business modeling is to provide a common modeling language for both communities. The business processes are documented using the so called business use–cases. This workflow is one point to connect the RUP and the ARIS concepts.

Requirements: The purpose of the requirements is to describe what the system should do. The developers and the customers should agree to the description, which is called the *vision document*. The actors/users and the use–cases are identified. These are documented in the *use–case model*. The use–case model is used as a baseline during the workflows *analysis & design* and *test*.

Non–functional requirements are described in the *supplementary specification*.

Analysis & Design: The purpose of the analysis & design workflow is to show how the system will be realized. The result is the design model. It is an abstraction of the source code and consists of design classes structured into design packages and design subsystems. It also contains descriptions of how objects of these design classes collaborate to perform the use–cases.

Implementation: The system is realized through implementation of components. It is described, how to use existing components and implementing new components in way, such they are reusable. Implementation languages are for example Java, C++, or Ada.

Test: The focus of the test workflow is to ensure the quality of the software. The purposes of testing are:

- To verify the integration between the objects
- To verify the proper integration of all components of the software.
- To verify that all requirements have been correctly implemented.
- To ensure that all defects are removed before the software is deployed.

As the RUP is an iterative process, in each iteration the different releases are tested. This allows to detect defects as early as possible, which radically reduces the cost of defects fixing.

Deployment: The purpose of the deployment workflow is to successfully deliver the software to its end users. This includes producing external releases of the software, packaging the software, distributing the software, installing the software, and providing help and assistance to the users.

Although deployment activities are mostly done in the transition phase, many preparation have been done in earlier phases.

Project management: In the RUP the project management workflow has three purposes:

- A framework for managing software–intensive projects.
- Practical guidelines for planning, staffing, executing, and monitoring projects.
- A framework for managing risk.

Some usual aspects as staff management, budget management, or contract negotiations, of project management are not considered in the RUP.

The planning of an RUP software project is done on two levels: *phases planning* and *iteration planning*. The plan of the phases is very rough, on the other hand the iteration plans are very detailed.

Configuration & Change management: In this workflow the control of numerous artefacts produced by the many people who work on the project are described. The following problems are considered: simultaneous update, overall notification of changes, version management.

Environment: The purpose of the environment workflow is to provide

- the software development tools
- configuration of the tools
- configuration of the process (cp. section 2.2.3)

- process improvement
- training
- technical support, administration, backups, etc.

Some activities are closely connected to the RUP configuration, which is described in the following.

2.2.3 RUP configuration

The RUP is such global, that it can be used without any configurations and changings. In particular this holds for small and middle software companies, without an explicit culture for the processes. On the other hand the RUP is a framework, which can be modified, adapted, and extended for special requirements of the company. If the RUP is configured, soonest the workers or the activities are changed.

3 Related work

In this section we present some papers about ARIS and UML. The goal of all approaches is to improve ARIS by integration of some object–oriented models. Nearly no paper attend to the process itself, as our approach in this paper.

As described in section 2.1 in [Sc01] the ARIS framework concept and its life cycle model is presented. This is a method to describe the building of business information systems. The approach is based on the traditional software development paradigms like the waterfall model, although in some views object–oriented models are integrated.

There are some papers, which give approaches to integrate the business views to object–oriented software development methods. Often these approaches describe the integration of Event–Driven Process Chains (EPC) into the Unified Modeling Language (UML) [FS00].

In [LF01] on the one hand the connections between the different workflows of the RUP and the EPCs are described. On the other hand a mapping between different UML diagrams and EPCs are given.

In [NTZ98] two approaches are described to combine *process* and *object–oriented* modeling. The first approach gives a transformation of business process models (EPCs) into object–oriented models. The UML models *use–case diagram*, *activity diagram*, and *class diagram* are considered.

The second approach summarizes the ideas of integration both methods by object–oriented event–driven chains (oEPC), which are introduced in [SNZ97]. Within oEPCs business objects and events respectively rules are defined as classes.

In [LA98] the base of the considerations is the ARIS framework concept, similar as in our considerations. Besides the regard of the connection between EPCs and the different UML diagrams, which build the basis of the considerations in [LF01], in this paper the statement is given that a straightforward sequence as in the waterfall software development process is not adequate to build a business information system. This statement is the base of our approach.

4 The ARIS Unified Information System Development Process

In this section we will extend the RUP, which is a method to develop iteratively a software system, to an iterative process developing business information systems. We call this process The ARIS Unified Information System Development Process (short ARIS Unified Process abbreviated as AUP). In our approach the rough ARIS process to model business information systems (figure 2) is developed to the iterative and incremental process AUP as the traditional waterfall model is developed to the iterative and incremental process RUP. As the development of a business information system contains software development, the RUP as described in section 2.2 is part of the AUP.

In this section we will describe the AUP in a similar way as we outlined the RUP above. After the overview, we will present the dynamic structure, which is given along the time in the phases *strategy*, *inception*, *elaboration*, *construction*, and *transition*. Then we give the static structure, which consists of the *workers*, the *activities*, the *artefacts*, and the *workflows*.

We will only consider the core process workflows as the core supporting workflows of the RUP are unchanged in the AUP.

Before we present the dynamic and the static structure, we will give an overview, which shows the two orthogonal structures.

In figure 5 the phases and the different workflows are presented. There are five main workflows. The first one is the strategy workflow. The complete activities of this workflow are done during the strategy phase. The other four workflows *requirements definition of control flow*, *requirements definition*, *design specification*, and *implementation* are derived from the rough ARIS process (figure 2). The workflows *requirements definition*, *design specification*, and *implementation* are divided into subworkflows. For each view of the ARIS–house there is respectively one subworkflow.

We close the overview by a description of the integration of the RUP into the AUP. In comparison to the RUP, the workflow *requirements control view* is comparable to the *business workflow*. The main difference are the assigned diagrams. As in the RUP business workflow, the business processes are modeled only by different use–case diagrams and business process object models, in the AUP the business ideas are modeled additionally by the EPCs. The subworkflow *requirements definition function view* and the first step of the subworkflow *data view*, the macro view, are comparable to the RUP workflow *requirements*. The subworkflow *design specification function view* and the subworkflow *control view* are comparable to the RUP *analysis & design* workflow.

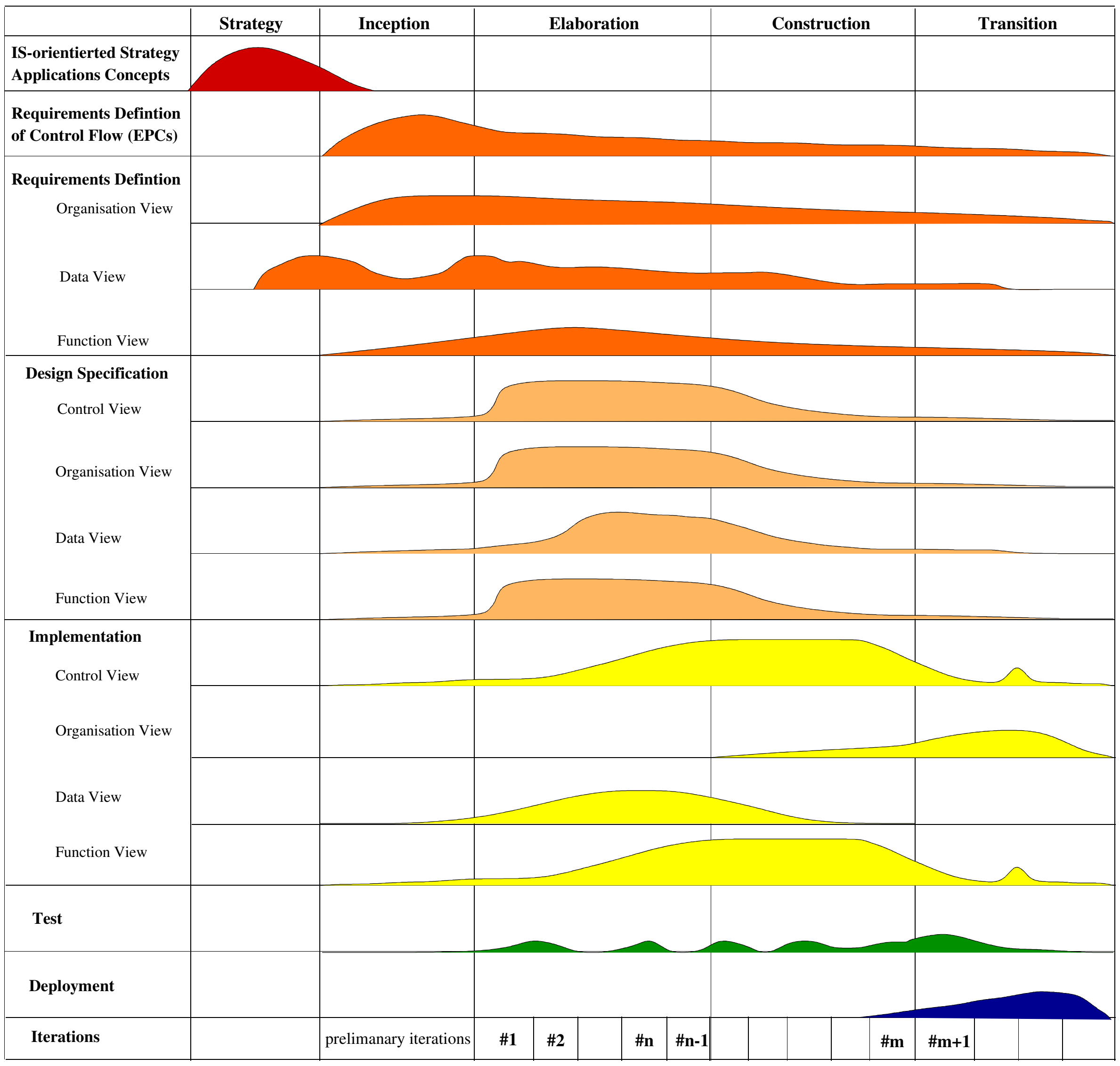


Figure 5: The ARIS Unified Process

4.1 The dynamic structure of the AUP– the phases

As in section 2.2, we will now consider the dynamic structure of the AUP first. An information system development project is divided in five phases. Each phase is again divided into several iterations as in the RUP. Each iteration is closed by a milestone and each phase is closed by a major milestone.

The phases are comparable to the RUP phases. In the AUP there is an additional phase, the *strategy phase*.

Strategy phase: The first phase of the information system development project is the strategy phase. In this phase only the *Information System–oriented Strategy Applications Concepts* workflow and the *Requirements data view* subworkflow are active. In the first workflow the management have thoughts about the company's strategy, which are influenced by the technical progress. The other active workflow determines the macro description of the used data which corresponds also to the general possibilities of the information technique. For example a phone call or fax are macro descriptions of data.

In these phases the framework of the business information system is determined.

Inception phase: The inception phase can be described very similar as in the RUP (cp. section 2.2.1). For the inception phase one principle is very important: It holds *structure follows process follows strategy* [OF96]. In the AUP this principle is applied iteratively. In the strategy phase the company's strategy is determined. Then, the *requirements definition of the control view* workflow (process) and the *organization* workflow (structure) form iteration by iteration the business model and the organization model.

The other phases *elaboration*, *construction*, and *transition* are not considered in this paper as they are similar to the respective phases of the RUP.

4.2 The Workflows

In the following we will describe the core workflows *Information System–oriented Strategy Applications Concepts*, *Requirements Definition of Control View*, *Requirements Definition*, *Design Specification*, *Implementation* and its subworkflows (fig. 5). The core supporting workflows from the RUP are unchanged in the AUP.

4.2.1 Information System–oriented Strategy Applications Concepts Workflow

This workflow is only active in the strategy phase. The workflow is described in the section of the strategy phase.

Workers: In this workflow the executive board of the company is responsible for the result of the workflow. During the workflow an analyzer of the information technique possibilities is involved.

Artefacts: The result of the workflow is a strategy glossary, which contains the strategy for the information system, which is to build.

4.2.2 Requirements Definition of Control View Workflow

The purposes of this workflow are to understand the structure and the processes of the company, to ensure that the customer, the user, and the developer have the same view of the company, and determine which requirements are necessary to support the company. This workflow is similar to the RUP *business workflow*.

The *requirements definition of control view* workflow is considered as an own workflow. In the other workflows the *control view* workflows are subworkflows. The reason is that the main importance of the control view here is a description of the whole information system in a global view. In the other workflows (*design specification*, *implementation*) the control view workflows are only necessary for the interface description of the different other views.

Worker:

- *Business process analyst*: The business process analyst analyses the consisting company, the structure and their processes. He specifies roughly the optimized processes and the following company's structure.
- *Business process designer*: The business process designer takes the roughly defined processes and describes the processes in detail. From this follows the detailed company's structure.

Artefacts: The results of this workflow are *use–case diagrams*, a mapping from a planning level diagram to different corresponding data (*data level diagram*), and different *EPCs*. The collection of these diagrams is called the *business model.*

The use–cases describe the rough processes and the organization units which are responsible for the respective functions of the processes. The use-cases can be modeled more detailed by state diagrams and sequence diagrams. The EPCs finally describe the process in detail. In the EPCs (modeled as extended EPCs) all views, organization, function, data, and product are connected. These models together are called the *use–case model.*

The allocation of the different data to the different planning levels are described in the *data level diagram*.

In comparison to the RUP approach the activity diagrams are not components of the use–case model, as the activity diagrams are less powerful than the EPCs.

In comparison to ARIS we do not model the connection between function and data view workflows by class diagrams at this point. We move this in the design specification workflow. The reason is, that it is often not possible for customers and stakeholders to understand class diagrams. Class diagrams are far away from the modeling techniques of business engineering.

This *requirements definition of control flow* workflow describes the connection between the three different views on the information system, which is to build. It is obvious, that this can only be done if the views themselves are also modeled. The iterative approach guarantees this, as the corresponding workflows are parallel active in the respective iterations.

4.2.3 Requirements Definition Workflow

The purpose of the requirements definition is to describe what the system should do and allows the developers and the customers to agree on that description.

The subworkflow

The workflow is divided into three subworkflows. Each subworkflow considers another view of the whole system respectively its environment.

- *Organization view*: The organization view workflow forms the structure of the company. This is done in close coordination to the company's strategy and the process definition.
- *Data view*: The data view workflow forms the macro and the micro description of the data structure. During the strategy phase the workflow starts by modeling the macro description as the macro data objects are defined in the strategy phase. At the end of the inception phase the modeling of the micro description starts.
- *Function view*: The function view workflow models the functionality of the information system.

Worker:

- *System analyst*: The system analyst controls and coordinates the requirements discovery. He gives the framework of the whole system and is responsible for the vision document.
- *Function designer*: The function designer specifies in detail all functions of the system.
- *Architect*: The architect is responsible for the identification of the use–cases and the functions, which influence the architecture.
- *Organization designer*: The organization designer is responsible for the advanced company's structure, which follows from strategy and process.
- *Data analyst*: The data analyst have to identify the different data representations of the information system. He is responsible for the macro data description.
- *Data designer*: The data designer takes the macro description and builds a data micro description, which contains the data structure of the information system.

Artefacts: The artefact, which is developed in all subworkflows is the *vision document*. The vision document contains the general vision of the core project's, key features, and the main constraints.

The artefacts of the subworkflow organization view are the *organigram* and the *diagram of the process-oriented planing levels*. These two diagrams form the *organization model*.

The artefacts of the data view subworkflow are a *model of the macro data description* and *ERM–diagrams* as a model of the micro data description.

Further artefacts are the *function hierarchies*, which are modeled in the function workflow.

4.2.4 Design Specification Workflow

The purpose of the design specification workflow is to show how the system will be realized in the implementation phase.

The subworkflows

This workflow is divided into four subworkflows. In the design specification the connection between the control view workflow and the other three workflows is closer than in the requirements definition. In the requirements definition the control view is a global view to the whole information system. Here the control view describes only the interfaces between the models of the different other views.

The main activities of the control view, the organization view, and the function view subworkflows start at the beginning of the elaboration phase. The data view subworkflow starts later, as the data view subworkflow of the requirements definition is longer active to model the ERM–diagrams.

- *Organization view*: In the organization view subworkflow a logical network is designed. The network is defined by its nodes and its edges. It is also modeled the rough capacity of the edges, a mapping of nodes and existing networks to organization units, and the protocols which are used.
- *Data view*: In the data view subworkflow the entity relationship model is transferred into statements of the data definition language SQL. This done in some steps as normalization, definition of views, and translation in SQL.
- *Function view*: In the function view subworkflow the algorithms of the system's functionality are determined. UML offers for this sequence diagrams, collaboration diagrams, activity diagrams, and state diagrams. This subworkflow has a close connection to the control view subworkflow, where the classes are modeled.
- *Control view*: In the control view workflow the interaction of the models of the other three subworkflows are built. The class diagrams are modeled as connection of the data and the function view subworkflows. But there is another problem. Often there is no obvious isomorph mapping from database tables to the classes, as the structures

of the database tables and the classes are completly different. Therefore a mapping from the class structure to the database schema must be developed.

The connection of the organization view and the data view workflow is described by a detailed authorization concept. The concept is a refinement of the mapping from data to planing levels of the requirements definition. In ARIS at this point there is additionally a description of distributed databases required. In the last years the capacity of networks arises enormously and the encryption techniques became more and more standard. Therefore in new designed systems databases are often hold at central points. In advanced systems the different data sources are made transparent by so-called application servers. This means that at least the collection of different data sources to an application server is to model at this point.

The connection between the organization view and the function view workflow is described by a mapping from network nodes to different program modules. Today it is often not necessary to consider this mapping as the program modules are hold central and the user interface is done by so-called thin-clients as web-clients.

The workers

- *Architect*: The architect controls and coordinates the technical artefacts. He is responsible for the matching of the different artefacts built in the different view subworkflows.
- *Class Designer*: The class designer is responsible for the realization of the use–case model and the building of the application server design. The realization is done by modeling of classes, packages, and subsystems.
- *Database Designer*: The database designer is responsible for the transfer of the ERM–diagrams to SQL–statements.
- *Data–Object-Translate Designer*: The data–object–translate designer is responsible for the mapping from the class structure to the database schema.
- *Function Designer*: The function designer is responsible for the deployment of the algorithms, which realizes the functions of the system.
- *Network Designer*: The network designer designs the logical network.
- *Authorization Designer*: The authorization designer determines the authorization of the different organization units to the different data.

For each of the designer there is also a reviewer, who reviews the artefacts, which are produced in the respective subworkflows.

The artefacts

One of the main artefacts is the *design model*. The design model consists of the class diagrams and the additional diagrams (object diagrams, state diagrams, sequence diagrams, collaboration diagrams, and activity diagrams) which describe the dynamic behavior of the class instances. The design model contains also the diagrams which are artefacts of the function view subworkflow. The mapping from the class structure to the database schema and the structure of a possibly given application server is also modeled in the design model. The design model is structured in subsystems and packages.

The *authorization concept* is the artefact which describe the authorization of organization units to work with different data.

The artefact which models the data view is the *database schema*. This is represented in the data definition language SQL.

In the organization view subworkflow the *logical network* is the modeled artefact.

4.2.5 The Implementation Workflow

The implementation workflow has five purposes:

- To define the organization of code in terms of subsystems.
- To implement classes and databases in terms of components.
- To test the developed components as units.
- To establish the network.
- To integrate into the executable information system the results of the implementers.

In the RUP three key concepts are introduced: *builds*, *integration*, and *prototypes*. These key concepts will be extended in the AUP.

Builds

A build is an operational version of the system or a part of the system that demonstrate a subset of the capabilities to be provided in the final product.

Builds in the AUP are parts of the implemented system, done in a programming language as well as parts of the database or parts of the network.

Integration

The integration is an activity in which separate software components, database components, or components of the network implementation are combined into a whole. In the AUP the integration is done incrementally, which means that the components are built in small pieces and then combined into a working whole by the addition of one piece at a time.

The incremental integration offers the following benefit: Faults are easier to locate, the components are tested more fully, and some parts of the system is running earlier.

Prototypes

Prototypes are used to reduce risk. Prototypes can reduce uncertainly surrounding the following issues: business viability, stability and performance of key technologies, understanding of the requirements, look and feel, and the usability.

The subworkflows

The three subworkflows *control view*, *data view*, and *function view* starts at a similar point. The activities in the data view subworkflow become earlier less, as by the result of the design specification, the SQL queries, the whole work is nearly done. In the other two subworkflows work is done nearly parallel, as the implementation of the classes is closely connected to the implementation of the functionality.

In the *organization view subworkflow* the network is established. The main work starts later as in the other subworkflows. The different nodes of the network can be simulated by only real existing computer. This is the reason, that the establishing of the network will start, when the databases and the implemented code is nearly finished.

The *test* workflow and the *deployment* workflow are not considered in this paper, as they are very similar to the respective workflows in the RUP. They must only be added by testing and deploying the databases and the network.

The workers

- *Architect*: The architect defines the structure of the implementation model (packages and subsystems)
- *Code Implementer*: The code implementer develops the separate components and tests them.
- *Database Implementer*: The database implementer implements the database components.
- *Network Builder*: The network builder transforms the logical network to physical one and establishes it.
- *System Integrator*: The system integrator constructs a build.

For each of the implementer respective the network builder there is also a reviewer, who reviews the artefacts, which are produced in the respective subworkflows.

The artefacts

The main artefacts of the implementation workflow are the *implementation subsystems*, the *components*, the *integration build plan*, and the *physical network*.

The implementation subsystems are a collection of components and other subsystems. A component is a source file, a binary, an executable, or a database component, or another file. The integration build plan specifies the builds to be created when the system is integrated. The physical network is plan of the network with real hardware components and its establishing.

5 Conclusion and outlook

In this paper we have developed the ARIS Unified Information System Development Process (AUP). The AUP arose from the Rational Unified Process (RUP) by extension of components which are defined in the ARIS framework concept. The ARIS framework concept considers besides the software development the construction of databases, networks, and its integration. In the ARIS framework concept an life cycle model is presented, which is similar to the well-known waterfall model in the software development. As the RUP is determined from the waterfall model by adding several iterations, the AUP is also determined from the ARIS life cycle by adding several iterations. The modeling language for the AUP is an aggregation of the UML models and the ARIS models.

The main benefit which we expect by using the AUP is to detect errors in the information system development earlier. This would reduce the costs of information system development enormously. We derive this assumption from the fact that RUP has reduced costs in pure software development projects. The main reason for earlier error detection is the close connection between customer and implementer.

This paper is an overview paper similar to [RUP] for the RUP. In the next time this paper is to advance to a full description of the AUP as it is done for example in [Kr03] for the RUP.

Furthermore we will build “AUP the product”. Analogous to RUP the product, it should offer some manuals and a configurable online version.

The modeling language is as said before an aggregation of the UML models and the ARIS models. The UML models are standardized in the look and feel. The ARIS models in contrast are only defined on the meta level by class diagrams. A further purpose is to define a unified language for the AUP, where the ARIS models satisfy the defined class diagrams on the meta level and the UML diagrams are given in version 2.0 [HJR$^+$04].

References

[Ba00] Balzert, H.: *Lehrbuch der Software–Technik*. Spektrum, Akademischer Verlag. 2. Auflage. 2000.

[FS00] Fowler, M. und Scott, K.: *UML konzentriert*. Addison Wesley Professional. 2000. (in german).

[HJR$^+$04] Hahn, J., Jeckle, M., Rupp, C., Zengler, B., und Queins, S.: *UML 2 glasklar*. Hanser Fachbuchverlag. 2004. (in german).

[JBR98] Jacobsen, I., Booch, G., und Rumbaugh, J.: *The Unified Software Development Process*. Reading, MA et al. 1998.

[KNS92] Keller, G., Nüttgens, M., und Scheer, A.-W.: Semantische Prozeßmodellierung. Technical Report 89. Institut für Wirtschaftsinformatik. Saarbrücken. 1992. (in german).

[Kr03] Kruchten, P.: *The Rational Unified Process: An Introduction*. Addison Wesley Professional. 3rd. 2003.

[LA98] Loos, P. und Allweyer, T.: Process Orientation and Object-Orientation – An Approach for Integrating UML and Event-Driven Process Chains (EPC). Technical Report paper 144. Instituts für Wirtschaftsinformatik. march 1998.

[LF01] Loos, P. und Fettke, P.: Towards an Integration of Business Process Modeling and Object-Oriented Software Development. In: *Proceedings of the Fifth International Symposium on Economic Informatics Bucharest*. S. 835–843. 2001.

[NTZ98] Nüttgens, M., Thomas, F., und Zimmermann, V.: Business Process Modeling with EPC and UML: Transformation or Integration? In: Schader, M. (Hrsg.), *The Unified Modeling Language - Technical Aspects and Applications, Proceedings (Mannheim, Oktober 1997)*. S. 250–261. Heidelberg. 1998.

[OF96] Osterloh, M. und Frost, J.: *Prozeßmanagement als Kernkompetenz, wie Sie Bussiness Reengineering stategisch nutzen können*. Wiesbaden. 1996. (in german).

[RUP] Rational Unified Process, Best Practices for Software Development Teams. A Rational Software Copporation White Paper.

[Sc01] Scheer, A.-W.: *ARIS - Modellierungsmethoden, Metamodelle, Anwendungen*. Springer–Verlag. 4. Auflage. 2001. (in german).

[Sc02] Scheer, A.-W.: *Vom Geschäftsprozeß zum Anwendungssystem*. Springer–Verlag. 4. Auflage. 2002. (in german).

[SNZ97] Scheer, A.-W., Nüttgens, M., und Zimmermann, V.: Objektorientierte Ereignisgesteuerte Prozeßkette (oEPK) - Methode und Anwendung. Technical Report Heft 141. Instituts für Wirtschaftsinformatik. 1997. (in german).