Construction of Process Models from Example Runs

Robin Bergenthum[1], Jörg Desel[1], Sebastian Mauser[1], Robert Lorenz[2] *

[1] Department of Applied Computer Science, Catholic University of Eichstätt-Ingolstadt,
Ostenstr. 14, 85072 Eichstätt, Germany
{robin.bergenthum,joerg.desel,sebastian.mauser}@ku-eichstaett.de
[2] Department of Computer Science, University of Augsburg,
Eichleitnerstrasse 30, 86159 Augsburg, Germany
robert.lorenz@informatik.uni-augsburg.de

Abstract. This contribution suggests a novel approach for a systematic and automatic generation of process models from example runs. The language used for process models is place/transition Petri nets, the language used for example runs is labelled partial orders. The approach adopts techniques from Petri net synthesis and from process mining. In addition to a formal treatment of the approach, a case study is presented and implementation issues are discussed.

1 Introduction

Business process modelling and management has attracted increasing attention in recent years [1–3]. However, little attention has been paid to the first phases of business process modelling, i.e., to the question of how to derive a valid process model in an informal setting.

The usual approach to process model construction and validation is shown on the left hand side in Figure 1. A domain expert edits a formal process model. Simulation tools generate single runs of that process model which can also be viewed as formal objects, such as occurrence sequences representing possible sequential occurrences of activities or occurrence nets representing occurrences of activities and their causal ordering. Then the expert checks whether these runs correspond to possible executions of the intended process. In the negative case, he changes the process model and iteratively repeats the simulation.

In this paper, we consider Petri net process models. There are many simulation tools that are able to generate sequential runs. Our VipTool generates and visualizes causally ordered occurrence nets [4, 5].

The aim of this paper is to suggest a proceeding in the opposite direction. We call causally ordered executions of the process to be modelled scenarios. We assume that the domain experts know some or all scenarios of the process to be modelled better than the process itself. Actually, experts might also know parts of the process model including parts of its branching structure, but in this case scenarios can be derived from this partially known process model. Experience shows that in various application areas processes are specified in terms of example scenarios (an evidence are the commonly used sequence diagrams in UML to specify scenarios).

* Supported by the project "SYNOPS" of the German Research Council

In a first step, an expert formalizes the scenarios, yielding formal runs. In other words, he provides formal models of the scenarios. In our setting, the scenarios are formalized in terms of labelled partial orders representing occurrences of process activities and their mutual order relation. In a second step, a process model is automatically generated from these formal runs. For this step, we apply algorithms developed for Petri net synthesis [6–8] and for process mining [9, 10]. This procedure is shown on the right hand side of Figure 1.

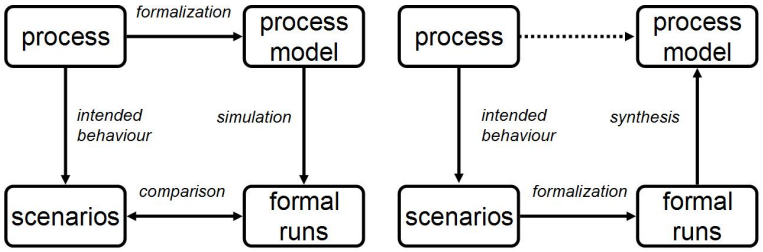

Fig. 1. Old and new approach.

The synthesized process model has at least the specified runs, but it might have additional runs. In a third step, additional runs are generated and presented to the expert. Runs that also represent legal scenarios are added to the set of runs specifying the process. If a run represents a behaviour which was not intended then the process model is changed accordingly.

In the following section we provide the necessary formalism to specify runs. The partial order approach taken in this paper avoids to consider all possible orderings of concurrent occurrences of activities as it was necessary in a sequential approach. However, in case of branching due to alternatives, the number of necessary example runs can be quite large (or even infinite). Therefore, we develop a term based specification of runs in section 3, where the atoms of the terms are comparably small labelled partial orders. Section 4 deals with the synthesis of a p/t-net process model from such term specification. In section 5 we tackle the problem of hierarchical process definitions. Section 6 provides a case study supported by our tool VipTool. In section 7 we discuss related work.

2 Specifying Runs

The core idea of our approach is to specify behaviour of a process in terms of single runs, playing the role of example runs of the process. Therefore, the process model to be generated should at least have the behaviour given by these runs.

We claim that modelling a single run is an easy and intuitive task, also for domain experts unexperienced in modelling. There are many possibilities to describe scenarios (also textual descriptions are adequate) and formalization of scenarios in terms of runs is relatively easy (on this level of single instances). Using scenarios in requirements engineering has received significant attention in the last years in the field of software modelling (see section 7). In this paper we focus on scenarios to model business processes. In this area, scenarios do not necessarily have to be designed from scratch. In

many cases it is possible to exploit already existing descriptions of scenarios supported by the business process. In an enterprise, typical sources of scenario descriptions are log files recorded by information systems (process mining focuses on this source of information), process instructions for employees or textual and formal process descriptions from some requirements analysis.

In the first step of the design approach, single runs of the process are identified. This leads to a preferably complete description of the behaviour of the process. In this paper, we consider *labelled partial orders (LPOs)* to specify single runs formally. LPOs are a very general formalism and most languages used in practice can be mapped to LPOs.

Definition 1 (labelled partial order). *A labelled partial order (LPO) is a triple* $\mathrm{lpo} = (V, <, l)$*, where* V *is a set of events,* $<$ *is an irreflexive and transitive binary relation on* V*, and* $l : V \to T$ *is a labelling function with set of labels* T*.*

The behaviour specified by an LPO includes its so called prefixes and sequentializations. An LPO $(V', <', l')$ *is called a prefix of another LPO* $(V, <, l)$ *if* $V' \subseteq V$*,* $(v' \in V' \wedge v < v') \Longrightarrow (v \in V')$*,* $<' = < \cap (V' \times V')$ *and* $l' = l|_{V'}$*. An LPO* $(V, <', l)$ *is called a sequentialization of another LPO* $(V, <, l)$ *if* $< \subseteq <'$*.*

Two LPOs $(V, <, l)$ *and* $(V', <', l')$ *are called isomorphic if there is a bijective mapping* $\psi : V \to V'$ *such that* $l(v) = l'(\psi(v))$ *for* $v \in V$*, and* $v < w \iff \psi(v) <' \psi(w)$ *for* $v, w \in V$*. Isomorphic LPOs model the same behaviour. Therefore, we consider LPOs only up to isomorphism, i.e., isomorphic LPOs are not distinguished.*

An LPO models a single run by specifying "earlier than"-dependencies between events, where an event represents an occurrence of the process activity given by its label. LPOs offer the following advantages in process modelling compared to sequential approaches where behaviour is given in terms of occurrence sequences [4]:

• *A natural and intuitive representation of the behaviour of processes:* Since concurrency plays an important role in process models, it is appropriate to model concurrency also in single runs of a process. In particular, instead of considering sequential runs and detecting the concurrency relation from a set of runs, it is easier and more intuitive to work with partially ordered runs.

• *An efficient representation of the behaviour of processes:* A single LPO represents a set of sequential runs, which can be quite large (exponential in the number of transition occurrences) in the presence of concurrency.

• *A high degree of expressiveness:* First, considering sequential runs, concurrency cannot be distinguished from non-deterministic resource sharing. Second, LPOs explicitly model causal dependencies between transition occurrences, which allows the explicit modelling of the flow of objects and of information in processes (this is not even implicitly possible with sequential runs).

• *Efficient analysis algorithms for business process models:* In many cases, analysis techniques applied to LPOs are more efficient than those working on sequential runs [5, 11].

We start our process generation procedure with a collection of LPOs (runs) representing scenarios of the process. An example of such a set of LPOs is shown in Figure 2 for the workflow triggered by a damage report in an insurance company. In all shown runs, a received claim is negatively evaluated and a refusal letter is sent. In the first run,

the refusal letter is sent after the damage and the insurance of the client was checked. In runs 2 and 3 only one check is performed before the negative evaluation and the sending of the refusal letter. In all runs, after the assignment and the registration of the claim, reserves are set aside.

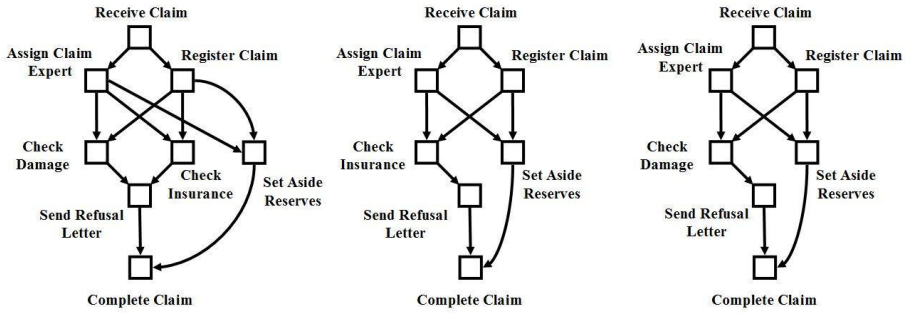

Fig. 2. Example for a collection of LPOs representing scenarios of a process.

Before formalizing scenarios, the domain expert should specify initial and final conditions for the considered process (see [12]) or some other kind of reference to the environment of the process in order to embed the process into its context. Moreover, he should identify the possible activities because they appear as labels in the runs. This is of particular importance when more than one expert provides example runs, because all occurrences of the same activity have to be labelled by the same activity name.

3 Composed Runs

In case of large processes the procedure of considering complete runs as described in the previous section may still be difficult for domain experts. In particular, it suffers from the fact that the number of runs that have to be specified might grow exponentially with the number of alternatives and can be even infinite if the process contains loops. Also, single example runs may become very large and difficult to handle. The problem can be solved by partly incorporating the process structure in the specification. The idea is that it is possible for a domain expert to only specify parts of runs, called run segments, which can be as small as desired and also can be used to locally specify alternative or iterative parts of runs (avoiding the explicit specification of all alternative runs). Complete runs are then given by specifying appropriate compositions of run segments. This possibility to modularly develop runs by means of run segments makes the specification of runs easier, faster and more intuitive for domain experts. Sometimes such approach is even necessary, e.g. for complete specifications in the case of iterations or if some domain expert actually only knows parts of runs.

Figure 3 depicts a run segment showing the registration process, several run segments describing possible evaluation procedures of the claim, a run segment modelling the payment of the insurance company and the three singleton runs for building reserves, gathering information for the payment by asking queries and the completion of the workflow. In this section, we specify a set of runs by means of such run segments.

Run segments can be related in four ways:

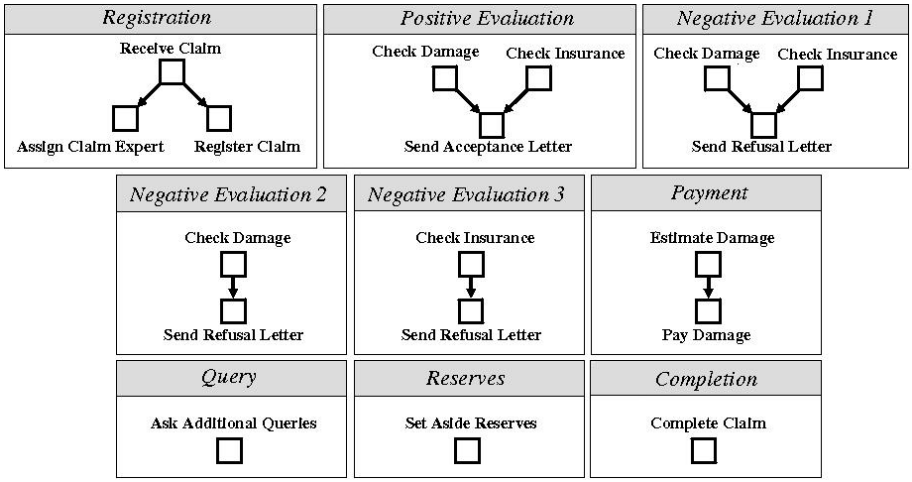

Fig. 3. Example for a collection of run segments in terms of LPOs.

- A run segment LPO_2 occurs after another segment LPO_1 (sequence).
- Either LPO_1 occurs or LPO_2 but not both (alternative).
- A run segment LPO_1 can recur arbitrarily often (iteration).
- Two segments LPO_1 and LPO_2 occur concurrently (concurrency).

Similar to the approach for scenario integration based on statecharts in [13], higher level structures of runs are built by concatenating and nesting blocks according to the relationships sequence (;), alternative ($+$), iteration ($*$), and concurrency ($||$). For the run segments of Figure 3, we assume that they are related as depicted in Figure 4 to faithfully model the underlying workflow. That means, the workflow starts with the "Registration" of the claim. Then one of the evaluation runs is performed concurrently to the subprocess of setting aside reserves for the claim. The four possibilities of evaluation are alternatives. The run modelling the positive case is a sequential composition of the three single run segments "Positive Evaluation", "Queries" and "Payment". Asking additional queries can be iterated arbitrarily often until a sufficient degree of information is reached. Finally, after the execution of all other run segments, the process is finished by the run segment "Completion". Figure 4 uses a graphical representation of the four composition templates for runs by means of a block structure (; -composition is depicted by arcs, $+$, $*$ and $||$ by respective symbols). Since the binary composition operators are associative, the readability of the graphical representation is improved by composing more than two blocks in figures (e.g. the $+$-composition of the four alternative evaluation possibilities). We call such a behavioural specification generated by the composition of single run segments a *composed run*. Composed runs nicely integrate modular scenario specifications (run segments) which may be given by different experts and support the specification of infinite behaviour (by the iteration operator).

Since single run segments are given by LPOs, the semantics of a composed run $\mathbf{R}$ is defined as a set of LPOs $\mathfrak{Lpo}(\mathbf{R})$ modelling possible behaviour each, called the *set of runs defined by a composed run*. The set of runs defined by the composed run depicted in Figure 4 comprises the runs shown in Figure 2 and the infinite set of runs illustrated in Figure 5 (activity "Ask Additional Queries" can be iterated).

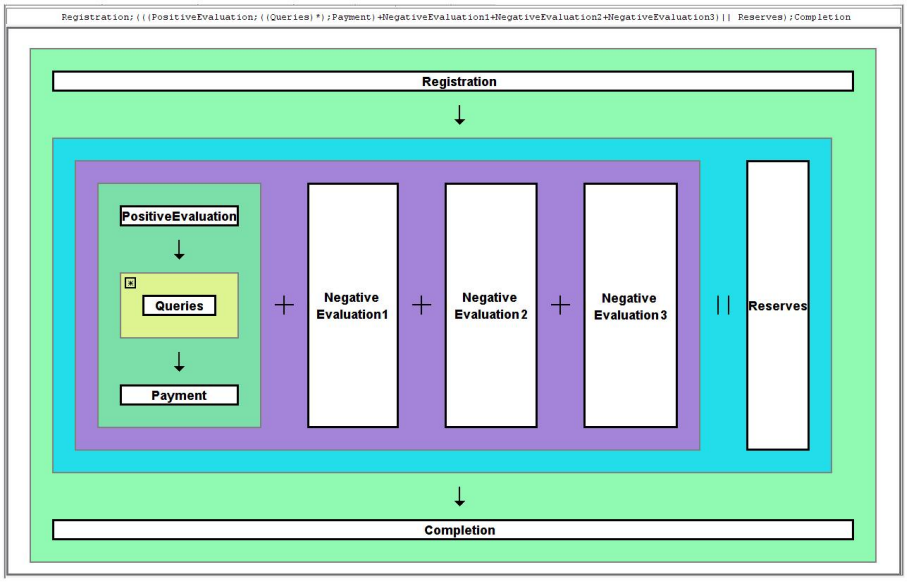

Fig. 4. A composed run over the set of run segments depicted in Figure 3 (screenshot of VipTool).

Definition 2 (composed run). *Given a finite set of single run segments $\mathcal{A}$, a composed run over $\mathcal{A}$ is inductively defined as follows:*
Each single run segment lpo *of $\mathcal{A}$ and the empty LPO $\lambda = (\emptyset, \emptyset, \emptyset)$ are composed runs. Let $\mathbf{R}_1$ and $\mathbf{R}_2$ be composed runs. Then $\mathbf{R}_1; \mathbf{R}_2$ (sequential composition), $\mathbf{R}_1 + \mathbf{R}_2$ (alternative composition), $(\mathbf{R}_1)^*$ (iteration) and $\mathbf{R}_1 \parallel \mathbf{R}_2$ (concurrent composition) are composed runs.*

Assume two LPOs $\mathrm{lpo}_1 = (V_1, <_1, l_1), \mathrm{lpo}_2 = (V_2, <_2, l_2)$ with disjoint sets of events. We define:

- $\mathrm{lpo}_1; \mathrm{lpo}_2 := (V_1 \cup V_2, <_1 \cup <_2 \cup (V_1 \times V_2), l_1 \cup l_2)$,
- $\mathrm{lpo}_1 \parallel \mathrm{lpo}_2 := (V_1 \cup V_2, <_1 \cup <_2, l_1 \cup l_2)$,
- $\mathrm{lpo}_1^0 := \lambda$ *and* $\mathrm{lpo}_1^n := \mathrm{lpo}_1^{n-1}; \mathrm{lpo}_1$ *for* $n > 0$.

The set of runs $\mathfrak{Lpo}(\mathbf{R})$ of a composed run $\mathbf{R}$ over $\mathcal{A}$ is a possibly infinite set of LPOs. Given a composed run $\mathbf{R}$, we first inductively define a set of LPOs $K(\mathbf{R})$ represented by $\mathbf{R}$. The set $\mathfrak{Lpo}(\mathbf{R})$ is the prefix and sequentialization closure of $K(\mathbf{R})$. We set $K(\lambda) = \{\lambda\}$ and $K(\mathrm{lpo}) = \{\mathrm{lpo}\}$ for $\mathrm{lpo} \in \mathcal{A}$. For composed runs $\mathbf{R}_1$ and $\mathbf{R}_2$,

- $K(\mathbf{R}_1 + \mathbf{R}_2) = K(\mathbf{R}_1) \cup K(\mathbf{R}_2)$,
- $K(\mathbf{R}_1; \mathbf{R}_2) = \{\mathrm{lpo}_1; \mathrm{lpo}_2 \mid \mathrm{lpo}_1 \in K(\mathbf{R}_1),\ \mathrm{lpo}_2 \in K(\mathbf{R}_2)\}$,
- $K((\mathbf{R}_1)^*) = \{\mathrm{lpo}_1; \ldots; \mathrm{lpo}_n \mid \mathrm{lpo}_1, \ldots, \mathrm{lpo}_n \in K(\mathbf{R}_1),\ n \in \mathbb{N}^+\} \cup \{\lambda\}$,
- $K(\mathbf{R}_1 \parallel \mathbf{R}_2) = \{\mathrm{lpo}_1 \parallel \mathrm{lpo}_2 \mid \mathrm{lpo}_1 \in K(\mathbf{R}_1),\ \mathrm{lpo}_2 \in K(\mathbf{R}_2)\}$.

A problem excluded so far is that some runs may overlap. That means, the knowledge about one run segment may be distributed on several experts, each knowing only a part of the segment. In the simplest case these parts can be treated as single run segments that can be composed as shown above. But this is not possible if the parts contain common events. Then they have to be fused to one single run segment. The situation of runs having common events occurs if several experts have different views to one

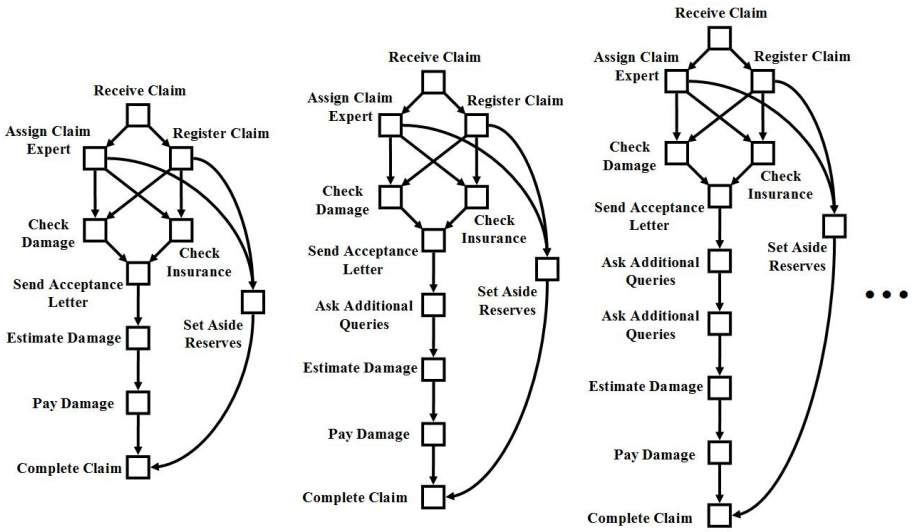

Fig. 5. Infinite set of runs of the composed run of Figure 4, where the claim is positively evaluated.

process execution, i.e. they observe different subsets of all events of the respective run, whereas respective other parts of the run are hidden.

We propose the following concept to *fuse run segments*. Given several parts of one segment, first the involved people have to determine which events observed by one expert coincide with which events observed by another expert. This problem has to be solved by an appropriate communication between the experts and is part of the specification process. The experts have to agree on a *fusion equivalence relation*, defined on the set of events of all parts of the considered run segment, such that different observations of one event are equivalent. Obviously, only events having the same label (referring to the same activity) can be equivalent. Also, the orderings given by different observations must not contradict each other. This has to be ensured in an adjustment phase by the modelers. The fusion of the parts is then given by a new LPO, which has an event for each equivalence class. If two events are ordered in some part of the run segment, then their respective classes are ordered in this LPO. Thus, each dependency observed (respectively modelled) by some expert is regarded in the fused run segment. No further dependencies are introduced. Conversely, we assume concurrency between events if no expert detected any dependency. This is motivated by the idea that parts of one run observed by different experts tend to be concurrent.

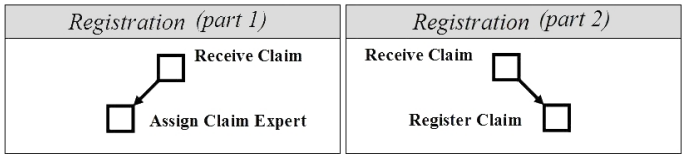

Fig. 6. Two possible parts of the run segment ”Registration” shown in Figure 3.

A simple example is shown in Figure 6. Assume that the run segment "Registration" of Figure 3 is not given directly, but rather by two parts of the run segment, as shown in Figure 6. If the two events labelled by "Receive Claim" coincide, the described fusion approach generates the fused run "Registration" of Figure 3.

Definition 3 (fusion). *Assume LPOs* $\mathrm{lpo}_i = (V_i, <_i, l_i)$, $i \in \{1, \ldots, n\}$ *with pairwise disjoint sets of events, modelling different parts of one run segment.*

An equivalence relation $\sim$ *on* $\bigcup_{i=1}^{n} V_i$ *fulfills the fusion requirement if*

$$v \sim v', v \neq v', v \in V_i, v' \in V_j \Longrightarrow$$

$$i \neq j \wedge l_i(v) = l_j(v') \wedge \forall v'' \in V_i, v''' \in V_j, v'' \sim v''' : (v <_i v'' \Longrightarrow v''' \not<_j v').$$

In this case, the fused LPO of $\mathrm{lpo}_i = (V_i, <_i, l_i)$, $i \in \{1, \ldots, n\}$ *w.r.t.* $\sim$ *is defined by* $\mathrm{lpo} = (V, <, l)$, *where*

- $V = \{[v]_\sim \mid v \in \bigcup_{i=1}^{n} V_i\}$,
- $[v]_\sim < [v']_\sim \Longleftrightarrow (\exists v'' \in [v]_\sim, v''' \in [v']_\sim, i \in \{1, \ldots, n\} : v'', v''' \in V_i, v'' <_i v''')$,
- $l([v]_\sim) = l_i(v)$ *(for* $v \in V_i$*).*

The fused LPO is well defined because of the fusion requirement.

4 Synthesizing a Process Model

The next step in the design approach starts with a specification of a process by means of a composed run. The aim is to automatically create a Petri net model from the composed run. Petri net based models are the standard to compactly represent processes in the area of workflow design and Petri nets offer a huge repertoire of analysis methods. Formally, the composed run is defined as a term over the alphabet of single run segments employing the composition operators $;$, $+$, $*$ and $||$. In [6], we show how to synthesize a *place/transition net (p/t-net)* from such a term.

The activities of the process are modelled by the transitions of the synthesized Petri net. The places together with their connections to the transitions and their markings define dependencies between the activities. As usual, places are drawn as circles, tokens in places represent the initial marking, transitions are drawn as rectangles and the flow relation as arcs annotated with values of the weight function (arcs with weight 0 are not drawn, the weight 1 is not shown). Note that events of a composed run having the same label model different occurrences of the same transition. Therefore, it is not possible to convert a composed run into a Petri net the naive way by adding places in between run segments and within run segments putting places in between ordered events.

A *run of a p/t-net* N is given by an LPO with event labels referring to transitions, such that the events can occur, respecting the concurrency and dependency relations of the LPO. Thus, a run describes executable behaviour of the net in the sense that the transition occurrences given by the events are possible in the net, only using the dependencies specified by the run for the flow of tokens. The set of all runs of N is denoted by $\mathfrak{Lpo}(N)$.

Definition 4 (p/t-net). *A (marked) p/t-net is a quadruple* $N = (P, T, W, m_0)$*, where* P *is a finite set of places,* T *is a finite set of transitions satisfying* $P \cap T = \emptyset$*,* $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ *is a weight function defining the flow relation, and* $m_0 : P \to \mathbb{N}$ *(*$\mathbb{N}$ *denotes the non-negative integers) is an initial marking.*

A multi-set of transitions $\tau : T \to \mathbb{N}$ is called a step *(of transitions). A step τ is enabled to occur (concurrently) in a marking $m : P \to \mathbb{N}$ of a p/t-net if and only if $m(p) \geq \sum_{t \in \tau} \tau(t) W(p,t)$ for each place $p \in P$. In this case, its occurrence leads to the marking $m'(p) = m(p) + \sum_{t \in \tau} \tau(t)(W(t,p) - W(p,t))$, abbreviated by $m \xrightarrow{\tau} m'$. A finite sequence of steps $\sigma = \tau_1 \dots \tau_n,\ n \in \mathbb{N}$, is called a step occurrence sequence enabled at m and leading to m_n, denoted by $m \xrightarrow{\sigma} m_n$, if there exists a sequence of markings $m_1, \dots, m_n$ such that $m \xrightarrow{\tau_1} m_1 \xrightarrow{\tau_2} \dots \xrightarrow{\tau_n} m_n$.*

Given an LPO $\mathrm{lpo} = (V, <, l)$, two events $v, v' \in V$ are called independent if $v \not< v'$ and $v' \not< v$, denoted by $v \,\mathrm{co}\, v'$. A co-set is a subset $C \subseteq V$ fulfilling: $\forall v, v' \in C : v \,\mathrm{co}\, v'$. A cut is a maximal co-set. For a co-set C and an event $v \in V \setminus C$ we write $v < (>)\, C$, if $v < (>)\, v'$ for an element $v' \in C$ and $v \,\mathrm{co}\, C$, if $v \,\mathrm{co}\, v'$ for all elements $v' \in C$. Given a marked p/t-net N, an LPO $\mathrm{lpo} = (V, <, l)$ with $l : V \to T$ is called a run of N if $m_0(p) + \sum_{v \in V \wedge v < C} (W(l(v), p) - W(p, l(v))) \geq \sum_{v \in C} W(p, l(v))$ for every cut C of lpo and every place p. The set of runs of a p/t-net N is defined by $\mathfrak{Lpo}(N) = \{(V, <, l) \mid (V, <, l)$ is a run of $N\}$.

A p/t-net N synthesized from a specified composed run $\mathbf{R}$ by the algorithm presented in [6] is a best upper approximation to $\mathbf{R}$ in the sense that

- $\mathfrak{Lpo}(\mathbf{R}) \subseteq \mathfrak{Lpo}(N)$ and
- $\forall (N') : (\mathfrak{Lpo}(\mathbf{R}) \subseteq \mathfrak{Lpo}(N')) \Longrightarrow (\mathfrak{Lpo}(N) \subseteq \mathfrak{Lpo}(N'))$.

Synthesizing an upper approximation is useful, because the behaviour explicitly specified by $\mathbf{R}$ should definitely be included in the behaviour of the synthesized model. The best upper approximation property ensures that only necessary additional behaviour is added to the synthesized net. Thus, computing a best upper approximation may be seen as a natural completion of the specified behaviour $\mathbf{R}$ by a Petri net.

After the generation of a process model in this way, in a follow-up validation step runs of the synthesized net which have not been specified are visualized. An expert is interactively asked whether these runs are legal or not. In the positive case, they are added to the specification. In the negative case, changes of the net to prohibit such runs are proposed to the expert. These changes always have the problem that the changed net does not anymore allow all specified runs. But it is possible to automatically compute such changes which prohibit a minimal number of specified runs. Additionally, since the specified behaviour is often incomplete and also the synthesized net tends to be incomplete, reasonable continuations of runs of the net are generated following certain heuristics and presented to the expert. The applied technique is deduced from the concept of wrong continuations [14, 8, 11]. Runs of the net are extended by appropriate events, and it is asked whether such additional runs model intended behaviour or not. To only propose a reasonable choice of such possible additional runs, different heuristic criteria such as considering runs occurring only once as a wrong continuation of the specified behaviour or runs prohibited by certain places can be applied. If such runs are desired, the net is changed accordingly and the runs are added to the set of runs specifying the process.

Finally, further heuristics to improve the readability of the net, as e.g. known from process mining [9], as well as partial order based validation techniques, as e.g. supported by VipTool [4], and verification techniques can be applied to further improve the process model.

5 Hierarchy

For large processes, knowledge about the process and its behaviour is often distributed in several involved people's minds. Some domain experts might have knowledge about the general process where single activities are on a high level of abstraction and have to be refined. Providing runs of this process leads to a corresponding model. Other people might know the behaviour of some details of the process, i.e. about the refinement of an activity of the main process, which defines a subprocess.

The paper [12] deals with synthesis of process models from this kind of distributed knowledge on process behaviour on different abstraction levels. Its results are mainly based on the observation that for partial order behaviour (in contrast to sequential order behaviour) subprocesses and the main process can first be synthesized independently and then be integrated. This section provides the core idea of this approach.

In the underlying design procedure, a special class of Petri nets is considered to model processes: connected p/t-nets with two distinguished sets of input and output transitions. In Figure 7 such nets are shown, where the input (output) transitions are depicted with two ingoing (outgoing) arcs.

Actually, in [12] no arc weights are considered. Moreover, the nets are required to have a certain 1-boundedness property (no reachable marking assigns more than one token to a place). The aim is that input and output transitions strictly alternate. This property ensures that the refinement step of the design procedure is correct. In [14], we show how 1-boundedness can be guaranteed by the considered synthesis algorithm. Nevertheless, in the present paper we consider the definition of processes based on general p/t-nets, because the synthesis algorithm is a lot more efficient in this case [15, 10, 8] (it is possible to apply fast standard linear programming techniques, while in the case of 1-boundedness integer linear programming methods are needed). To still allow the *refinement* procedure for transitions from [12], it has to be ensured that never a second instance of some subprocess is started (by an initial transition) before a prior instance of the same subprocess is finished (by a final transition). This can be achieved by adding to a transition t, before it is refined, a self-loop place p_t with $W(p_t, t) = W(t, p_t) = m_0(p_t) = 1$. Refining t by a subprocess, the place p_t has an outgoing arc with weight one to every input transition of the subprocess and an ingoing arc with weight one from every output transition of the subprocess (see Figure 7). Now we can guarantee the desired property by requiring that in the subprocess extended by p_t, input and output transitions can only occur strictly alternatingly, i.e. it is not possible that two input transitions occur without an intermediate output transition and vice versa. To avoid deadlocks, it additionally has to be required that from each reachable marking of the subprocess extended by p_t, there is a firing sequence including an output transition.

Formally, the refinement steps in our setting are defined as follows: We consider one main process. A subprocess refines a transition (to which a self-loop place was added) which either belongs to the main process or to another subprocess. It replaces the transition, the transition's input places are connected to the input transitions of the subprocess with arcs having the same weights, whereas the output transitions are connected to the output places of the transition by arcs having the same weights (see Figure 7). If we require the above behavioural restrictions for the subprocess, the external behaviour of the subprocess resembles the behaviour of the transition, with the difference that first

the input tokens are consumed and later the output tokens are produced. That means, the behaviour of the main process is preserved when refinement is applied. The order of refinements does not matter.

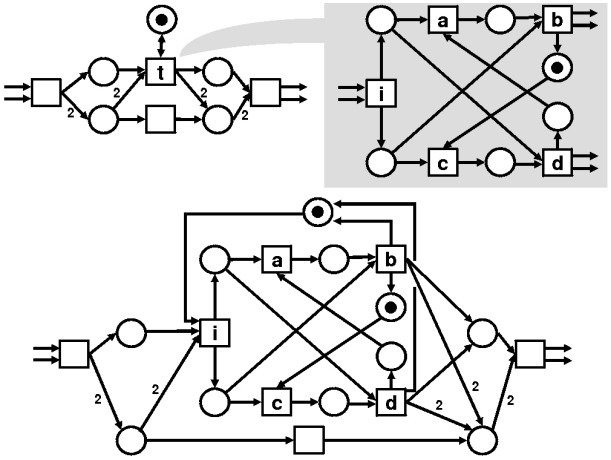

Fig. 7. Top: Two abstract processes. Bottom: The refinement of the left process w.r.t. t and the right process.

Definition 5 (refinement). *A process net is a connected p/t-net (P, T, W, m_0) with two distinguished sets of input and output transitions $T_i, T_o \subseteq T$.*

Let $N = (P, T, W, m_0)$ be a process net with a transition t and let $N^t = (P^t, T^t, W^t, m_0^t)$ be a process net with input transitions T_i^t and output transitions T_o^t. Assume w.l.o.g. that the elements of P and of P^t as well as of T and of T^t are disjoint. The refinement of N w.r.t. transition t and process N^t is defined as $(P \cup P^t, T \cup T^t \setminus \{t\}, W_{N,N^t}, m_0 \cup m_0^t)$, where W_{N,N^t} is defined by

$$W_{N,N^t}(x,y) = \begin{cases} W(x,y) & \text{if } x,y \in P \cup T \setminus \{t\}, \\ W^t(x,y) & \text{if } x,y \in P^t \cup T^t, \\ W(x,t) & \text{if } x \in {\bullet t} \wedge y \in T_i^t, \\ W(t,y) & \text{if } x \in T_0^t \wedge y \in t{\bullet}, \\ 0 & \text{otherwise .} \end{cases}$$

It only remains to tune the considered synthesis algorithm to the definition of process nets. The sets of input and output transitions have to be regarded in the synthesis algorithm to create reasonable process models.

Therefore, first the sets of input transitions T_i and output transitions T_o have to be defined. To ensure that a process starts with an input transition, finishes with an output transition and in between no input and output transitions occur, we require for the specification given by the composed run $\mathbf{R}$ that

• every non-empty LPO $(V, <, l)$ in the set of runs $\mathfrak{Lpo}(\mathbf{R})$ of $\mathbf{R}$ contains exactly one event $v_0 \in V$ (called initial event) labelled by an input transition of the process ($l(v_0) \in T_i$),

• the initial event v_0 of every non-empty LPO $(V, <, l) \in \mathfrak{Lpo}(\mathbf{R})$ is a unique minimal

event, i.e. it fulfills $v_0 < v$ for every $v \in V \setminus \{v_0\}$,
• every LPO $(V, <, l)$ in the set of runs $\mathfrak{Lpo}(\mathbf{R})$ of $\mathbf{R}$ which is not prefix of another LPO in $\mathfrak{Lpo}(\mathbf{R})$, called maximal LPO of $\mathfrak{Lpo}(\mathbf{R})$, contains exactly one event $v_{max} \in V$ (called final event) labelled by an output transition of the process ($l(v_{max}) \in T_o$),
• the final event v_{max} of every maximal LPO $(V, <, l) \in \mathfrak{Lpo}(\mathbf{R})$ is a unique maximal event, i.e. it fulfills $v < v_{max}$ for every $v \in V \setminus \{v_{max}\}$.
With these requirements, it is ensured that a net synthesized from $\mathbf{R}^*$ ($*$ is considered because a subprocess can be invoked arbitrarily often) fulfills the above behavioural requirements for subprocesses.

In the running example of the business process of an insurance company, the only input transition is "Receive Claim" and the only output transition is "Complete Claim". Therefore, the formulated requirement is fulfilled by our example composed run shown in Figure 4.

Figure 7 shows an example of a subprocess net refining a transition of a main process. Notice that this subprocess is equipped with a memory feature: In the first invocation of the subprocess, only the sequence cd is executable, in a second invocation only ab is possible, in a third one again cd, and so on. This memory feature was not possible if we would expect the subprocess to start with the empty marking, as it is the case for workflow nets [1].

6 Case Study and Tool Support

In this section we briefly illustrate the proposed synthesis procedure by the small case study which was already used for examples. We recently implemented appropriate synthesis features into our Petri net toolset VipTool, which offers a flexible xml-based open plug-in architecture. New plug-ins of VipTool allow to graphically specify a composed run and then to automatically synthesize a net from such specification. The refinement aspects of the business process design procedure of [12] are not yet supported by VipTool. Thus we consider the generation of a main process to show the applicability of the implemented synthesis algorithm.

The new editing functionalities for composed runs allow to compose LPOs drawn with other plug-ins of VipTool by respective composition operators. The composition of LPOs is graphically supported by a visualization of composed runs by block structures as introduced in this paper (compare Figure 4). The new editor also offers an alternative visualization in the form of UML activity diagrams (see Figure 8). The new algorithm to synthesize a net from a composed run is described in [16]. It combines the idea of wrong continuations of LPOs [14, 11], which proved to yield nice synthesis results, with the notion of regions introduced in [6]. A synthesized net can be loaded, visualized, layouted, edited and analyzed by other plug-ins of VipTool.

Figure 8 shows the specification of the running example depicted as an activity diagram in VipTool. The possible runs of this workflow have been depicted in Figures 2 and 5. The workflow starts by receiving a claim submitted by a client, followed by two concurrent activities "Assign Claim Expert" and "Register Claim" (segment "Registration"). The first one models the assignment of a claim expert in charge for this claim, the latter is concerned with the registration of the client and the loss form. Then, con-

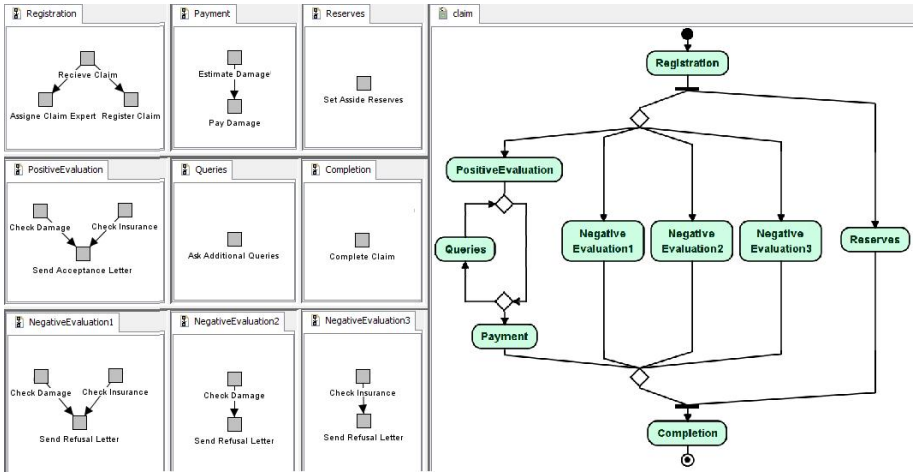

Fig. 8. Screenshot of VipTool showing the composed run from Figure 4 represented as an activity diagram and the corresponding run segments drawn in VipTool.

currently reserves for the claim are established (segment ”Reserves”) and the evaluation of the claim is started. The evaluation comprises four alternative runs. Each begins with two concurrent activities ”Check Damage” and ”Check Insurance”. ”Check Insurance” represents checking validity of the clients insurance, ”Check Damage” models checking of the damage itself. The segment ”Positive Evaluation” models the situation that both checks are evaluated positively, meaning that an acceptance letter is sent after the two checks. If one evaluation is negative, the company sends a refusal letter. Thus, the activity ”Send Refusal Letter” is performed after the two ”Check” activities if one is evaluated negatively (segment ”Negative Evaluation 1”). If a negative evaluation of one ”Check” activity already causes sending a refusal letter, while the other ”Check” activity has not been performed yet, this second ”Check” activity has to be disabled (i.e. it does not occur in a respective run), since it is no more necessary (segments ”Negative Evaluation 2” and ”Negative Evaluation 3”). In the case of a positive evaluation, either the damage is immediately estimated and payed (segment ”Payment”), or before the damage is estimated additional queries to improve estimation of the loss (segment ”Queries”) are repeatedly asked until sufficient information is collected. If the evaluation of the claim (including possibly paying the damage) and the segment ”Reserves” are finished, the process can be completed by the segment ”Completion”.

Figure 9 shows the net automatically created with the synthesis algorithm of VipTool from the specification depicted in Figure 8 (actually the synthesis algorithm generated two more places which have been deleted by a plug-in of VipTool searching for implicit places). Although the net seems to be complex on the first glance, it represents a very appealing model of the described business process. In fact, the net exactly has the specified behaviour (no additional, not specified runs) and there is no more compact possibility to describe the complex control flow of this business process by a Petri net.

The example illustrates that directly designing a Petri net model of a business process is often challenging, while modelling runs and synthesizing a net is more easy.

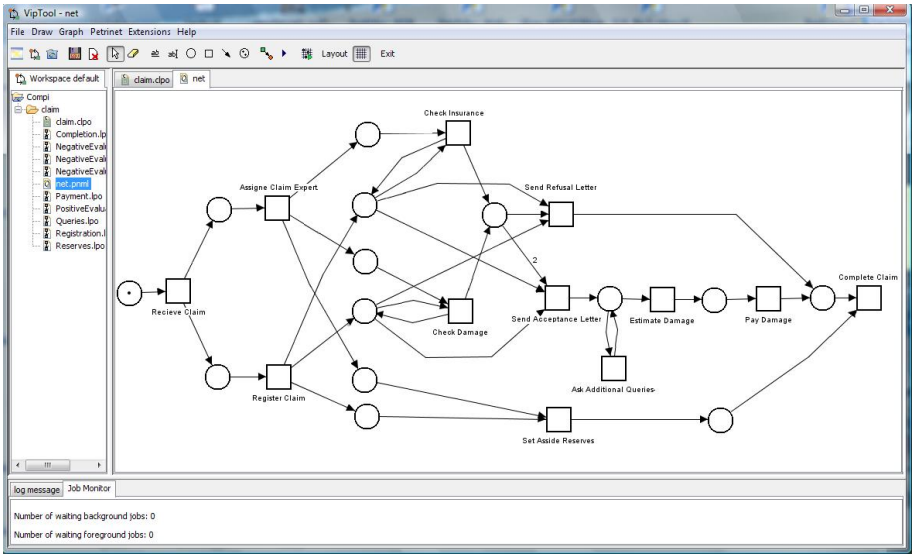

Fig. 9. Screenshot of VipTool showing the user interface of the editor for Petri nets.

Manually developing a complex Petri net such as our example net for the described workflow is an error-prone task.

7 Discussion

In the field of software engineering, the main approaches to system modelling have been structured analysis and structured design, developed in the late 1970's, as well as object-oriented analysis and design, starting in the late 1980's [17]. In the 1990's it was recognized on a broad front that requirements engineering – the elicitation, documentation and validation of requirements - is a fundamental aspect of software development and requirements engineering emerged as a field of study in its own right. Scenarios, firstly introduced by Jacobson's use cases [18], proved to be a key concept for writing system requirements specifications. Important advantages of using scenarios in requirements engineering include the view of the system from the viewpoint of users, the possibility to write partial specifications, the ease of understanding, short feedback cycles and the possibilities to directly derive test cases [19, 20]. Modelling software systems by means of scenarios received much attention over the past years. The dozens of popular scenario notations including e.g. the ITU standard of Message Sequence Charts, UML Sequence Diagrams, UML Communication Diagrams, UML Activity Diagrams and UML Interaction Overview Diagrams as well as Live Sequence Charts, are an evidence for this development. Several methodologies to bridge the gap between the scenario view of a system and state-based system models, which are closer to design and implementation, have been proposed [21, 22]. There are analytic, fully-automated synthetic and interactive synthetic software engineering methods to construct design models from scenarios. In a more radical approach [17] it is even suggested that a scenario specification may be considered not just the system's requirements but actually its final implementation.

In this paper we suggested focusing on scenarios to design business processes. Looking at scenarios to specify the requirements of a business process has similar advantages as in the software engineering domain. We developed a comprehensive methodology to model business processes which on the one hand regards the specifics of business processes and on the other hand exploits the benefits of scenario modelling. The approach starts by specifying example scenarios of the process in terms of (composed) runs and establishing respective necessary preconditions and relationships. Then a fully-automated synthesis algorithm generates a Petri net model of the process from the example scenarios which is afterwards interactively adjusted. In the domain of business processes modelling such construction methodology of process models focusing on example scenarios is an innovative approach. Most of the methods known from software engineering are not suited for business process design, because there are several differences which have to be regarded. E.g., in software modelling [17, 22, 21] the focus is on components or objects, communication (dependencies) between components and the distinction between inter- and intra-object behaviour, while in business process modelling [1–3] the emphasis is on global activities, dependencies through pre- and post-conditions of activities, and resources for activities. Modularity comes into play by appropriate refinement and composition concepts. In business process modelling so far (partially ordered) scenarios have only rarely been used and their application was restricted to analysis of process models, e.g. [4, 23]. An exception, where scenarios are directly used in the design phase, is process mining [9]. But usually process mining is very much adjusted to scenarios of event logs and restricted to sequential scenarios. Nevertheless, there is one approach, called multi-phase mining [24–26], where the final phase [25] algorithmically generates a process model from a finite collection of arbitrary partially ordered scenarios given e.g. by instance EPCs [25] or message sequence charts [26] (by directly translating dependencies from the scenarios to the process model). In contrast to this approach, we build our modelling methodology on formal methods known from Petri net synthesis [8, 6]. The advantage compared to [24, 25] is that our approach generates reliable results for all kinds of specifications (also in the presence of very complex routing structures). However, our techniques may be inferior w.r.t. performance issues.

8 Future Research

Results from practical application and evaluation will be important for further development of our methodology. The suggested approach to design business processes is based on several assumptions, but we believe that it could be helpful in many cases. However, this research is still in an initial phase and we do not have experiences from real applications. Future work includes defining of success criteria and empirical research. In particular, it would be interesting to identify and characterize settings in which our approach is superior to other approaches.

References

1. Aalst, W., Hee, K.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge, Massachsetts (2002)

2. Aalst, W., Dumas, M., Hofstede, A.: Process-Aware Information Systems – Bridging People and Software. Wiley (2005)
3. Weske, M.: Business Process Management – Concepts, Languages and Architectures. Springer (2007)
4. Desel, J., Juhás, G., Lorenz, R., Neumair, C.: Modelling and Validation with Viptool. In: BPM 2003, LNCS 2678, Springer (2003) 380–389
5. Bergenthum, R., Desel, J., Juhás, G., Lorenz, R.: Can I Execute My Scenario in Your Net? Viptool Tells You! In: Petri Nets 2006, LNCS 4024, Springer (2006) 381–390
6. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of Petri Nets from Infinite Partial Languages. In: ACSD 2008, IEEE (2008) 170–179
7. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of Petri Nets from Finite Partial Languages. Fundam. Inform. (to appear)
8. Badouel, E., Darondeau, P.: Theory of Regions. In: Advanced Course: Petri Nets 1996, LNCS 1491, Springer (1996) 529–586
9. Aalst, W.: Finding Structure in Unstructured Processes: The Case for Process Mining. In: ACSD 2007, IEEE (2007) 3–12
10. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process Mining Based on Regions of Languages. In: BPM 2007, LNCS 4714, Springer (2007) 375–383
11. Bergenthum, R., Mauser, S.: Comparison of Different Algorithms to Synthesize a Petri Net from a Partial Language. In: Workshop CHINA, Petri Nets 2008, X'ian (2008)
12. Desel, J.: From Human Knowledge to Process Models. In: UNISCON 2008, LNBIP 5, Springer (2008) 84–95
13. Glinz, M.: An Integrated Formal Model of Scenarios Based on Statecharts. In: ESEC 1995, LNCS 989, Springer (1995) 254–271
14. Lorenz, R., Juhás, G., Mauser, S.: How to Synthesize Nets from Languages - a Survey. In: Wintersimulation Conference 2007, Washington (2007) 638–647
15. Lorenz, R., Bergenthum, R., Desel, J., Mauser, S.: Synthesis of Petri Nets from Finite Partial Languages. In: ACSD 2007, IEEE (2007) 157–166
16. Bergenthum, R., Mauser, S.: Synthesis of Petri Nets from Infinite Partial Languages with VipTool. In: AWPN 2008, Rostock (2008) 81–86
17. Harel, D., Marelly, R.: Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer (2003)
18. Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley (1992)
19. Glinz, M., Seybold, C., Meier, S.: Simulation-Driven Creation, Validation and Evolution of Behavioral Requirements Models. In: MBEES 2007, Dagstuhl (2007) 103–112
20. Glinz, M.: Improving the Quality of Requirements with Scenarios. In: Second World Congress on Software Quality, Yokohama (2000) 55–60
21. Amyot, D., Eberlein, A.: An Evaluation of Scenario Notations and Construction Approaches for Telecommunication Systems Development. Telecommunication Systems **24**(1) (2003) 61–94
22. Liang, H., Dingel, J., Diskin, Z.: A Comparative Survey of Scenario-Based to State-Based Model Synthesis Approaches. In: SCESM 2006, ACM (2006) 5–12
23. Scheer: IDS Scheer: ARIS Process Performance Manager. http://www.ids-scheer.com.
24. Dongen, B., Aalst, W.: Multi-Phase Process Mining: Building Instance Graphs. In: Conceptual Modeling - ER 2004, LNCS 3288, Springer (2004) 362–376
25. Dongen, B., Aalst, W.: Multi-Phase Process Mining: Aggregating Instance Graphs into EPC's and Petri Nets. In: 2nd Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management, Petri Nets 2005, Miami (2005) 35–58
26. Lassen, K., Dongen, B., Aalst, W.: Translating Message Sequence Charts to Other Process Languages Using Process Mining. In: PNSE 2007, Petri Nets 2007, Siedlce (2007) 82–97