Conformance Checking of Processes Based on Monitoring Real Behavior

A. Rozinat and W.M.P. van der Aalst

Group of Information Systems, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
{a.rozinat,w.m.p.v.d.aalst}@tue.nl

Abstract. Many companies have adopted Process-aware Information Systems (PAIS) to support their business processes in some form. On the one hand these systems typically log events (e.g., in transaction logs or audit trails) related to the actual business process executions. On the other hand explicit process models describing how the business process should (or is expected to) be executed are frequently available. Together with the data recorded in the log, this situation raises the interesting question "Do the model and the log *conform* to each other?". Conformance checking, also referred to as conformance analysis, aims at the detection of inconsistencies between a process model and its corresponding execution log, and their quantification by the formation of metrics. This paper proposes an incremental approach to check the conformance of a process model and an event log. First of all, the *fitness* between the log and the model is measured (i.e., "Does the observed process comply with the control flow specified by the process model?"). Second, the *appropriateness* of the model can be analyzed with respect to the log (i.e., "Does the model describe the observed process in a suitable way?"). Appropriateness can be evaluated from both a *structural* and a *behavioral* perspective. To operationalize the ideas presented in this paper a *Conformance Checker* has been implemented within the ProM framework, and it has been evaluated using artificial and real-life event logs.

1 Introduction

New legislation such as the Sarbanes-Oxley (SOX) Act [33] and increased emphasis on corporate governance and operational efficiency have triggered the need for improved auditing systems. To audit an organization, business activities need to be monitored. Buzzwords such as BAM (Business Activity Monitoring), BOM (Business Operations Management), BPI (Business Process Intelligence) illustrate the interest of vendors to support the monitoring and analysis of business activities. The close monitoring of processes can be seen as a second wave following the wave of business process modeling and simulation. In the first wave the emphasis was on constructing process models and analyzing them, illustrated by the many notations available (e.g., Petri nets, UML activity diagrams, EPCs, IDEF, BPMN, and not to mention the vendor or system specific notations). This development has created the interesting situation where processes are being monitored while at the same time there are process models describing these processes. The focus of this paper is on *conformance*, i.e., "Is there a good match between the recorded events and the model?". A term that could be used in this context is "business alignment", i.e., are the real process (reflected by the log) and the process model (e.g., used to configure the system) aligned properly. Consider also Figure 1, where conformance checking is positioned in the broader context of process mining techniques. While *discovery* aims at the automatic extraction of a process model from log data, conformance checking is concerned with the comparison of an existing process model and a corresponding log. As soon as one is confident in the conformance of the model and the log, *extension* techniques can be used to project diagnostic information derived from the log onto the model (for example, to visualize performance bottlenecks in the process model).

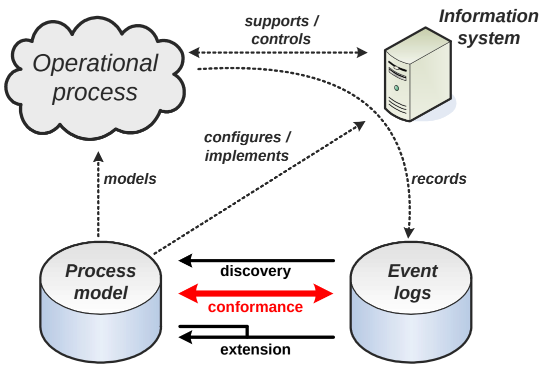


Fig. 1. Conformance checking in the broader context of process mining

Most information systems, e.g., WFM, ERP, CRM, SCM, and B2B systems, provide some kind of *event log* (also referred to as transaction log or audit trail) [9]. Typically such an event log registers the start and/or completion of activities. Every event refers to a case (i.e., process instance) and an activity, and, in most systems, also a timestamp, a performer, and some additional data. In this paper, we only use the first two attributes of an event, i.e., the identity of the case and the name of the activity. Meanwhile, any organization documents its processes in some form. The reasons for making these process models are manifold. Process models are used for communication, ISO 9000 certification, system configuration, analysis, simulation, etc. A process model may be of a *descriptive* or of a *prescriptive* nature. Descriptive models try to capture existing processes without being normative. As an example, in a hospital process it must be possible to react to urgent situations and, therefore, the flexibility to diverge from the normal flow of actions is crucial. Another example could be a model that was made to document a certain procedure in a financial system (which logs events of the activities that were executed *without* being driven by an explicit process model). Clearly, it is desirable to keep this model aligned with the actual procedure in the financial system using regular conformance checking techniques. Prescriptive models describe the way that processes should be executed. In a Workflow Management (WFM) system prescriptive models are used to enforce a particular way of working using IT [5]. However, as shown in one of the case studies presented later in this paper, users may need to deviate even if they work with prescriptive models in a rigid WFM system. Furthermore, in most situations prescriptive models are not used directly by the information system. For example, the reference models in the context of SAP R/3 [23] and ARIS [34] describe the "preferred" way processes should be executed. People actually using SAP R/3 may deviate from these reference models. Finally, even if the process model and the event log are fully compliant, it is often interesting to see how frequent certain parts in the model are actually used, and to potentially remove obsolete parts which otherwise would need to be maintained.

In this paper, we will use Petri nets [17] to model processes. Although the metrics are based on the Petri net approach, the results of this paper can be applied to any modeling language that can be equipped with executable semantics. An event log is represented by a set of event sequences, also referred to as traces. Each case in the log refers to one sequence. The most dominant requirement for conformance is **fitness**. An event log and Petri net "fit" if the Petri net can generate each trace in the log. In other words: the Petri net should be able to "parse" every event sequence. We will show that it is possible to quantify fitness, e.g., an event log and Petri net may have a fitness of 0.66. Unfortunately, a good fitness does not imply conformance. As we will show, it is easy to construct Petri nets that are able to parse any event log. Although such Petri nets have a fitness of 1 they do not provide meaningful information. Therefore, we introduce a second dimension: **appropriateness**. Appropriateness tries to capture the idea of *Occam's razor*, i.e., "one should not increase, beyond what is necessary, the number of entities required to explain anything". Clearly, this dimension is not as easy to quantify as fitness. We will distinguish between *structural appropriateness* (if a simple model can explain the log, why choose a complicated one) and *behavioral appropriateness* (the model should not be too generic and allow for too much behavior). Using examples, we will show that both the structural and behavioral aspects need to be considered to measure appropriateness adequately.

To actually measure conformance, we have developed a tool called *Conformance Checker*. It is part of the *ProM framework*[1], which offers a wide range of tools related to process mining, i.e., extracting information from event logs [9].

This paper extends an earlier paper on conformance [32]. In this paper we give explicit definitions of metrics, present new metrics, describe the implementation and present applications of the approach. The remainder of the paper is organized as follows. Section 2 introduces a running example that will be used to illustrate the concept of conformance, and provides the preliminaries that are needed to understand the concepts introduced later on. Section 3 discusses the

[1] Both documentation and software (including the source code) can be downloaded from *http://www.processmining.org*.

need for two dimensions. The fitness dimension is described in Section 4, the appropriateness dimension is elaborated in Section 5, and Section 6 evaluates how these dimensions can be combined. Next, Section 7 shows how these properties can be verified using the Conformance Checker in ProM, and discusses some implementation details. Then, Section 8 describes two different applications of the presented conformance checking techniques. Finally, some related work is discussed, and the paper is concluded.

2 Preliminaries

This section introduces a running example that will be used to illustrate the concept of conformance, and explains three basic concepts that are needed to understand the conformance checking techniques defined later in this paper, namely Petri nets (Section 2.1), event logs (Section 2.2), and the mapping between them (Section 2.3).

2.1 Petri Nets

The example process used throughout the paper concerns the processing of a liability claim within an insurance company. Figure 2 shows a *Petri net* [17] model of this liability claim handling procedure. It sketches a fictive (but possible real-world) procedure and exhibits typical control flow constructs that are relevant in the context of conformance checking.

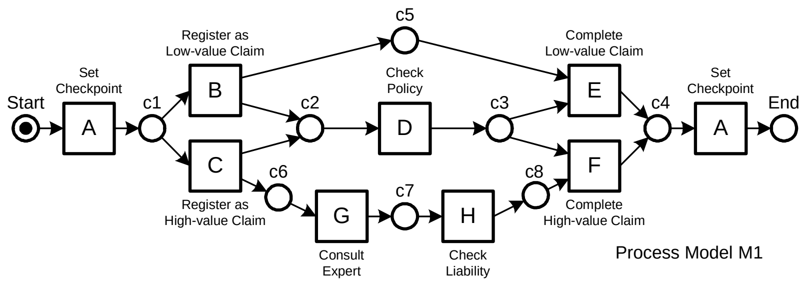


Fig. 2. Simplified model of processing a liability insurance claim

A Petri net is a dynamic structure that consists of a set of *transitions*, which are indicated by boxes and relate to some task, or action that can be executed, a set of *places*, which are indicated by circles and may hold one or more *tokens* (indicated as black dots), and a set of *directed arcs* that connect these transitions and places with each other in a bipartite manner. Transitions are *enabled* as soon as all of their input places (places connected to this transition via an incoming arc) contain a token. If a transition is enabled, it may *fire* whereas it consumes a token from each of its input places and produces a token for each of its output places (places connected to this transition via an outgoing arc). This way, the firing of a transition may change the *marking* of a net, and therefore the state of the process, which is defined by the distribution of tokens over the places. In the following we explain how the Petri net model in Figure 2 can be interpreted.

At first, there are two tasks bearing the same label "Set Checkpoint" (we use A as a shorthand to refer to this label). This can be thought of as an automatic backup action within the context of a transactional system, i.e., activity A is carried out at the beginning to define a rollback point enabling atomicity of the whole process, and at the end to ensure durability of the results. Then the actual business process is started with the distinction of low-value claims and high-value claims, which get registered differently (B or C). The policy of the client is always checked (D) but in the case of a high-value claim, additionally, the consultation of an expert takes place (G), and then the filed liability claim is being checked in more detail (H). The two completion tasks E and F can be thought of as two different sub-processes involving decision making and potential payment, taking place in another department. Note that the choice between E and F is influenced by the former choice between B and C (i.e., the model does not belong to the class of free-choice nets [16]).

In the remainder of this paper we assume that each process model belongs to a well-investigated subclass of Petri nets that is typically used to model business processes, which is the class of *sound WF-nets* [5]. A WF-net requires the Petri net to have (i) a single *Start* place, (ii) a single *End* place, and (iii) every node must be on some path from *Start* to *End*, i.e., the process is expected to define a dedicated begin and end point and there should be no "dangling" tasks in between. The soundness property further requires that (iv) each task can be potentially executed (i.e., there are no dead tasks), and (v) that the process—with only a single token in the *Start* place—can always terminate properly (i.e., finish with only a single token in the *End* place). Note that the soundness property guarantees the absence of deadlocks and live-locks.

2.2 Event Logs

We assume that process executions are recorded in an *event log*. Events in the log are only expected to (i) refer to an activity from the business process, (ii) refer to a case (i.e., process instance), and (iii) be totally ordered. Figure 3 shows three example logs for the process described in Figure 2 at an aggregate level. This means that process instances (i.e., sequences of log events grouped according to the Case ID) that exhibit the same event sequence are combined as a logical log trace, which stores the number of combined instances to weigh the importance of each trace. This is possible because only the control flow perspective is considered here. In a different setting like, e.g., mining social networks [8], the resources performing an activity would distinguish those instances from each other.

Note that none of the logs contains the sequence $ACGHDFA$, although the Petri net model would allow this. In fact it is highly probable that a log does not exhibit all possible sequences, since, e.g., the duration of activities or the

No. of Instances	Log Traces
4070	ABDEA
245	ACDGHFA
56	ACGDHFA

(a) Event Log L1

No. of Instances	Log Traces
1207	ABDEA
145	ACDGHFA
56	ACGDHFA
23	ACHDFA
28	ACDHFA

(b) Event Log L2

No. of Instances	Log Traces
24	BDE
7	AABHF
15	CHF
6	ADBE
1	ACBGDFAA
8	ABEDA

(c) Event Log L3

Fig. 3. Three logs for the process described in Figure 2: *No. of Instances* indicates the frequency, and *Log Traces* the actual event sequence for each trace. For example: event log *L1* contains 4070 cases following the sequence *ABDEA*, i.e., first *A* (“Set Checkpoint”) is executed, then *B*, etc.

availability of suitable resources may render some sequences very unlikely to occur. With respect to the example model one could think of task *D* as a standard task that can be performed by anyone in a short time period and task *G* and *H* as highly specialized and time-consuming checks, so that finishing *G* and *H* before *D* would be possible but practically may not happen. Note that, furthermore, the number of possible sequences generated by a process model may grow exponentially, in particular for a model containing concurrent behavior. For example, there are $5! = 120$ possible combinations for executing five tasks, and $8! = 40320$ for executing eight tasks that are parallel to each other. Therefore, an event log cannot be expected to exhibit *all* possible sequences of the underlying behavioral model. Process mining techniques strive for weakening the notion of *completeness*, i.e., the amount of information a log needs to contain to be able to rediscover the underlying process model [11].

2.3 Mapping

A prerequisite for conformance analysis is that the tasks in the process model must be associated with the logged events, which we represent by a label denoting the associated log event type (if any) for each task in the model. Besides the simple 1-to-1 mapping, where a task is associated with exactly one type of log event and no other task in the model is associated with the same type of log event, a mapping may result in the following constructs:

Duplicate Tasks Multiple tasks in the model are associated with the same type of log event, which means that they may be different but their occurrence cannot be distinguished in the log. Note that duplicate tasks only emerge from the mapping, since the tasks of a process model themselves *are* distinguishable (be it not by means of their label but their identity, or unique position in the graph). In the example process model in Figure 2 there are two tasks that bear the same label “Set Checkpoint”. They are duplicate tasks.

Invisible Tasks Tasks are not logged and, therefore, have no log event associated (in contrast, tasks that are logged are called visible tasks). This can happen because certain steps in the process might not be observable (such as a telephone call). Invisible tasks can also be introduced for routing purposes. In the remainder of this paper invisible tasks are denoted as small tasks filled with black color.

Note that, although activities in real business scenarios are often logged at a more fine-grained level—they record, for example, the scheduling, the start, and the completion of an activity—we assume that a task is associated to at most one type of log event (typically this corresponds to a *complete*[2] event) to keep the approach as universal as possible. We assume further that all log events that are not associated to any task in the model are removed before starting the analysis. In principle, this is a reasonable choice as the log could be at a different level of granularity than the model (e.g., contain not only *complete* but also *start* events, or error codes and system status messages), and we deliberately want to abstract from these more low-level events. Note that, however, this might be problematic in the context of adaptive workflow management systems such as ADEPT [30], where, for example, additional activities can be inserted for a process instance. More research is needed to specifically consider conformance in adaptive workflow management.

However, we can define the following auxiliary metrics, which indicate the degree of overlap from a log-based and a model-based perspective, respectively.

Metric 1 (Log Coverage) *Given a set of log entries E, a set of tasks T, and a set of labels L, let $l_E \in E \rightarrow L$, $l_T \in T \not\rightarrow L$, $T_V = dom(l_T)$, $L_T = \{l_T(t) \mid t \in T_V\}$, and $L_E = \{l_E(e) \mid e \in E\}$. The log coverage metrics c_E and c_{LE} are defined as follows:*

$$c_E = \frac{|\{e \in E \mid l_E(e) \in L_T\}|}{|E|} \tag{1}$$

$$c_{LE} = \frac{|L_E \cap L_T|}{|L_E|} \tag{2}$$

Note that because not every task in the model needs to be associated with a label (invisible tasks are unlabeled), the mapping between tasks and labels is represented by a partial function ($\not\rightarrow$). Therefore, the subset of tasks that *are* labeled (i.e., they are in the domain of l_T) is precisely the set of visible tasks (i.e., $T_V = dom(l_T)$). L_E and L_T denote the set of labels covered by the log and the model, respectively.

If we assume that $|E| > 0$ and $|L_E| > 0$, then these metrics range from 0 (in the case that none of the log entries is associated to any task in the model) to 1 (when every log entry is associated to at least one task in the model). Note that the metric c_E really quantifies the overlap on the log entry level, i.e., if, for example, only one type of log entry is not covered by the model but it

[2] The life cycle of an activity has been standardized by the MXML format for workflow logs, which is used by the ProM framework (refer to *http://www.processmining.org* for further information and the schema definition).

happens very often in the log, then this metric will reflect this, whereas metric c_{LE} measures the degree of overlap with respect to the types of log entries only.

Metric 2 (Model Coverage) *Given a set of log entries E, a set of tasks T, and a set of labels L, let $l_E \in E \rightarrow L$, $l_T \in T \not\rightarrow L$, $T_V = dom(l_T)$, $L_T = \{l_T(t) \mid t \in T_V\}$, and $L_E = \{l_E(e) \mid e \in E\}$. The model coverage metrics c_T and c_{LT} are defined as follows:*

$$c_T = \frac{|\{t \in T_V \mid l_T(t) \in L_E\}|}{|T_V|} \tag{3}$$

$$c_{LT} = \frac{|L_T \cap L_E|}{|L_T|} \tag{4}$$

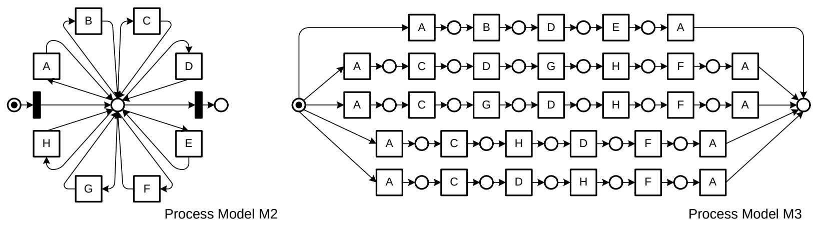
If we assume that $|T_V| > 0$ and $|L_T| > 0$, then these metrics range from 0 (in the case that every visible task in the model did not occur at all in the log) to 1 (in the case that each visible task occurred at least once in the log). Note that the metric c_T quantifies the overlap on the task (or transition) level, whereas metric c_{LT} measures the degree of overlap with respect to the different types of task labels only (i.e., it is abstracted from duplicate tasks).

3 Two Dimensions of Conformance: Fitness and Appropriateness

The most dominant question in the context of conformance is whether the real business process complies with the specified behavior, i.e., whether the log *fits* the model. With respect to the example model *M1* in Figure 2 this seems to apply for event log *L1*, since every log trace can be associated with a valid path from *Start* to *End*. In contrast, event log *L2* does not match completely because the traces *ACHDFA* and *ACDHFA* lack the execution of activity *G*, while event log *L3* does not even contain one trace corresponding to the specified behavior. Somehow, *L3* seems to fit "worse" than *L2*, and we want to measure the degree of fitness according to this intuitive notion of conformance.

But there is another interesting—rather qualitative—dimension of conformance, which can be illustrated by relating event log *L2* to the process models *M2* and *M3*, which are shown in Figure 4(a) and Figure 4(b). Although the log fits well with respect to both models, i.e., the event streams of the log and the model can be matched perfectly, they do not seem to be *appropriate* in describing the insurance claim administration.

The first model is much too generic as it covers a lot of extra behavior; it allows for arbitrary sequences containing the activities *A*, *B*, *C*, *D*, *E*, *F*, *G*, or *H*. The latter does not allow for more sequences than those that were observed in the log, but it only lists the possible sequences instead of expressing the specified behavior in a meaningful way. Therefore, it does not offer a better understanding than can be obtained by just looking at the aggregated log. We claim that a "good" process model should somehow be minimal in structure to clearly reflect



(a) Workflow model on a too high level of abstraction (i.e., too generic)

(b) Workflow model on a too low level of abstraction (i.e., too specific)

Fig. 4. Fitting models do not need to be a good representation of the observed behavior

the described behavior, in the following referred to as *structural appropriateness*, and minimal in behavior to represent as closely as possible what actually takes place, which will be called *behavioral appropriateness.*

As we have seen, conformance checking demands for two different types of metrics, which are:

- *Fitness*, i.e., the extent to which the log traces can be associated with valid execution paths specified by the process model, and
- *Appropriateness*, i.e., the degree of accuracy in which the process model describes the observed behavior, combined with the degree of clarity in which it is represented.

In the next two sections our goal is to develop conformance checking techniques that enable a business analyst to both (a) *measure* these two dimensions of conformance and (b) *locate* potential points of improvement:

(a) Metrics are important to estimate the severity of potential deviations, and to compare different model-log combinations with each other. Therefore, we want to define these metrics so that they are *stable*, i.e., as little as possible affected by properties that are not relevant, and *analyzable*, which in general relates to the scale type of a metric (such as nominal, ordinal, interval, or ratio scale). Note that, for example, the existence of a ratio scale enables propositions like model A is "twice as appropriate" as model B (with respect to a certain log) rather than only saying that the first model is "better" than the second [24]. However, an in-depth scale type discussion of the presented metrics is beyond the scope of this paper. In the context of conformance checking, it is especially desirable that the value of a metric indicates whether there is room for improvement, i.e., that it reaches some optimal value as soon as no better "match" would be possible.

(b) The localization of errors is crucial as otherwise it is not possible to gain more insight into a problem, and to issue potential alignment actions.

Note that a perceived conformance problem can always be *viewed from two angles.* First of all, the model may be assumed to be "correct" because it represents the way the business process should be carried out. Therefore, it might, for example, trigger actions to enforce the specified behavior. Second, the event log may be assumed to be "correct" because it is what really happened, and the process model might be either outdated or just not tailored to the needs of the employees actually performing the tasks. Highlighting this issue facilitates the redesign of the model and therefore increases transparency. In any case, a final interpretation can only be given by a domain expert. But even if the model and the log *do* conform to each other, this can be an important insight as it increases the confidence in the existing process model. A model validated through conformance checking may be the starting point for other types of analysis. Moreover, quantitative data extracted from the log may be projected on the model (e.g., frequencies, probabilities, bottlenecks, etc.).

4 Measuring Fitness

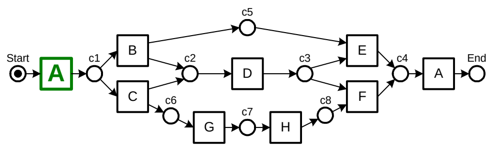
One way to measure the fit between event logs and process models is to replay the log in the model and somehow measure the mismatch, which subsequently is described in more detail. The replay of every logical log trace starts with the marking of the initial place in the model. Then, the transitions that belong to the logged events in the trace are fired one after another. While replay progresses, we count the number of tokens that had to be created artificially (i.e., the transition belonging to the logged event was not enabled and therefore could not be *successfully executed*) and the number of tokens that were left in the model, which indicate that the process was not *properly completed*.

Metric 3 (Fitness) *Let k be the number of* different *traces from the aggregated log. For each log trace i ($1 \leq i \leq k$), n_i is the number of process instances combined into the current trace, m_i is the number of missing tokens, r_i is the number of remaining tokens, c_i is the number of consumed tokens, and p_i is the number of produced tokens during log replay of the current trace. The token-based fitness metric f is defined as follows:*

$$f = \frac{1}{2}\left(1 - \frac{\sum_{i=1}^{k} n_i m_i}{\sum_{i=1}^{k} n_i c_i}\right) + \frac{1}{2}\left(1 - \frac{\sum_{i=1}^{k} n_i r_i}{\sum_{i=1}^{k} n_i p_i}\right)$$

Note that, for all i, $m_i \leq c_i$ and $r_i \leq p_i$, and therefore $0 \leq f \leq 1$. Note also that c_i and p_i cannot be 0 because during log replay there will be always at least one token produced for the *Start* place and one token consumed from the *End* place. To have a closer look at the log replay procedure consider Figure 5, which depicts the replay of the first trace from event log *L2* in process model *M1*. At the beginning (a) one initial token is produced for the *Start* place of the model. Initially, $m = 0$ (no missing tokens), $r = 0$ (no remaining tokens), $c = 0$ (no consumed tokens), and $p = 1$ (prior to the execution of A a token is put into place *Start*). The first log event in the trace, A, is associated with two transitions in the model each bearing the label A. But only one of them is enabled and thus will be fired (b), consuming the token from *Start* and producing one token for place *c1* ($c = 1, p = 2$). For the next log event the corresponding

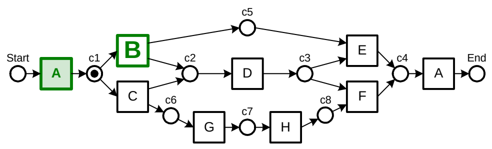
(a)



m = 0
r = 0
c = 0
p = 1

No. of Instances	Log Traces
1207	→ ABDEA
145	ACDGHFA
56	ACGDHFA
23	ACHDFA
28	ACDHFA

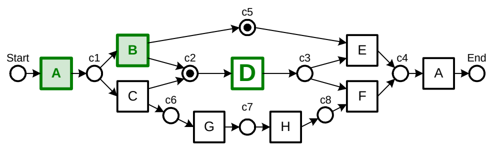
(b)



m = 0
r = 0
c = 1
p = 2

No. of Instances	Log Traces
1207	→ ABDEA
145	ACDGHFA
56	ACGDHFA
23	ACHDFA
28	ACDHFA

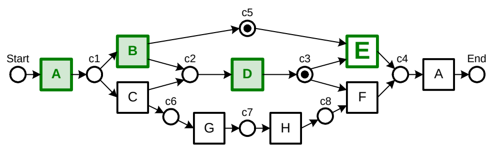
(c)



m = 0
r = 0
c = 2
p = 4

No. of Instances	Log Traces
1207	→ ABDEA
145	ACDGHFA
56	ACGDHFA
23	ACHDFA
28	ACDHFA

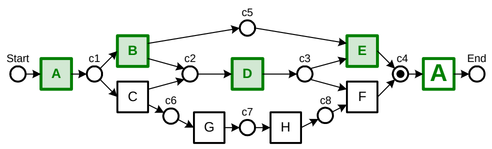
(d)



m = 0
r = 0
c = 3
p = 5

No. of Instances	Log Traces
1207	→ ABDEA
145	ACDGHFA
56	ACGDHFA
23	ACHDFA
28	ACDHFA

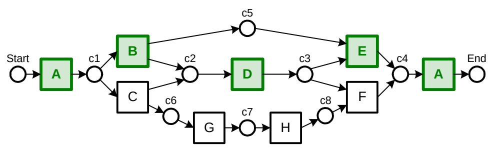
(e)



m = 0
r = 0
c = 5
p = 6

No. of Instances	Log Traces
1207	→ ABDEA
145	ACDGHFA
56	ACGDHFA
23	ACHDFA
28	ACDHFA

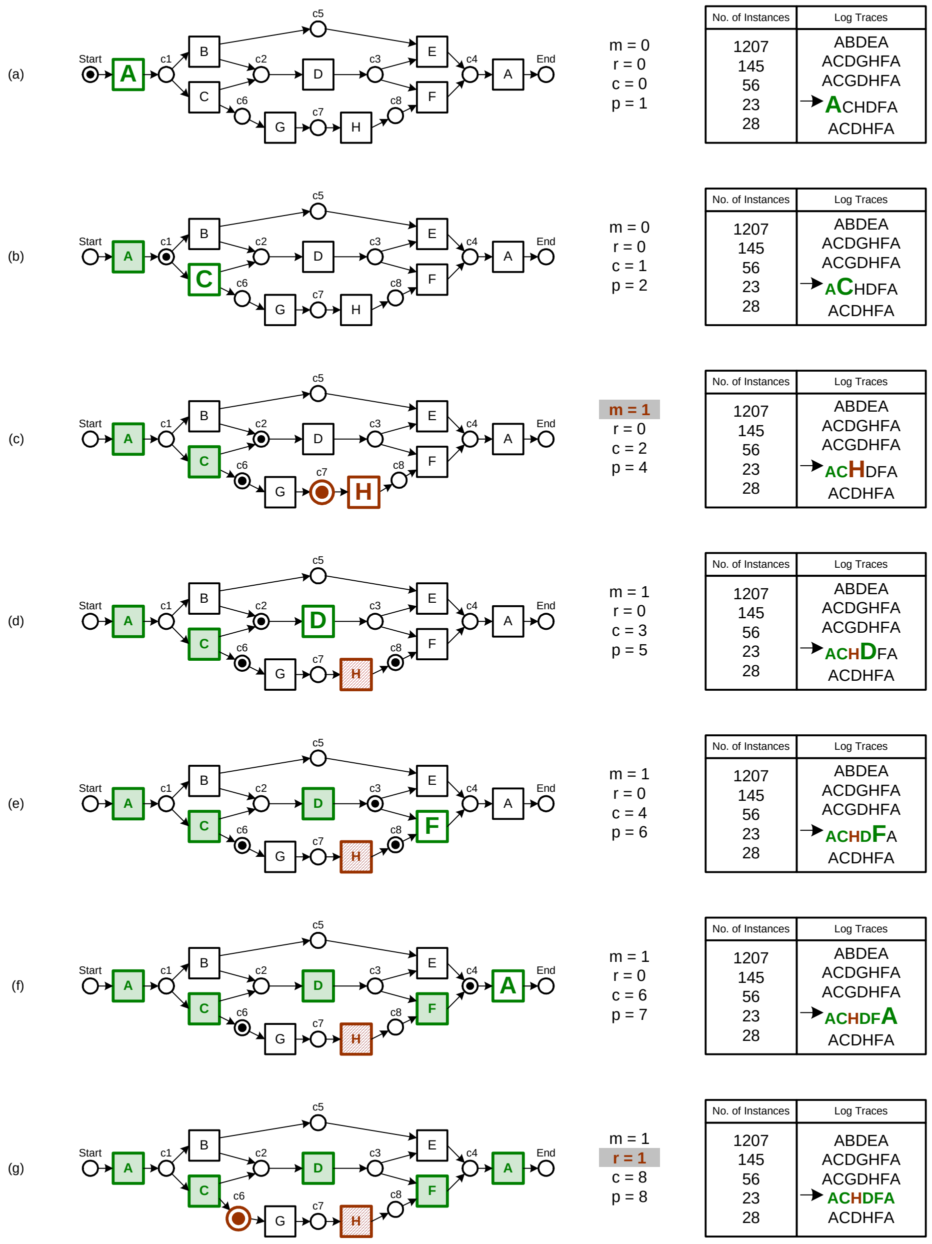
(f)



m = 0
r = 0
c = 7
p = 7

No. of Instances	Log Traces
1207	→ ABDEA
145	ACDGHFA
56	ACGDHFA
23	ACHDFA
28	ACDHFA

Fig. 5. Log replay for trace $i = 1$ of event log $L2$ in process model $M1$. The trace can be replayed without any problems, i.e., no tokens are missing ($m = 0$) or remaining ($r = 0$)



(a)

m = 0
r = 0
c = 0
p = 1

No. of Instances	Log Traces
1207	ABDEA
145	ACDGHFA
56	ACGDHFA
23	→ ACHDFA
28	ACDHFA

(b)

m = 0
r = 0
c = 1
p = 2

No. of Instances	Log Traces
1207	ABDEA
145	ACDGHFA
56	ACGDHFA
23	→ ACHDFA
28	ACDHFA

(c)

m = 1
r = 0
c = 2
p = 4

No. of Instances	Log Traces
1207	ABDEA
145	ACDGHFA
56	ACGDHFA
23	→ ACHDFA
28	ACDHFA

(d)

m = 1
r = 0
c = 3
p = 5

No. of Instances	Log Traces
1207	ABDEA
145	ACDGHFA
56	ACGDHFA
23	→ ACHDFA
28	ACDHFA

(e)

m = 1
r = 0
c = 4
p = 6

No. of Instances	Log Traces
1207	ABDEA
145	ACDGHFA
56	ACGDHFA
23	→ ACHDFA
28	ACDHFA

(f)

m = 1
r = 0
c = 6
p = 7

No. of Instances	Log Traces
1207	ABDEA
145	ACDGHFA
56	ACGDHFA
23	→ ACHDFA
28	ACDHFA

(g)

m = 1
r = 1
c = 8
p = 8

No. of Instances	Log Traces
1207	ABDEA
145	ACDGHFA
56	ACGDHFA
23	→ ACHDFA
28	ACDHFA

Fig. 6. Log replay for trace $i = 4$ of event log *L2* in process model *M1*. The replay of this trace requires the artificial creation of one token ($m = 1$) and one token is left behind ($r = 1$)

transition B is enabled and can be fired (c), consuming the token from *c1* and producing one token both for *c2* and *c5* $(c = 2, p = 4)$. Then, the following log event corresponds to transition D, which is enabled and therefore can be fired (d), consuming the token from *c2* and producing a token for *c3* $(c = 3, p = 5)$. Similarly, the transition associated to the next log event E is also enabled and fires (e), consuming the token from *c3* and *c5*, and producing one token for *c4* $(c = 5, p = 6)$. Finally, the last log event is of type A again, i.e., is associated with the two transitions A in the model. But only one of them is enabled and therefore chosen to be fired (f), consuming the token from *c4* and producing one token for the *End* place $(c = 6, p = 7)$. As a last step, this token at the *End* place is consumed $(c = 7)$ and the replay for that trace is completed, i.e., the removal of the token from *End* is seen as a consumption. Because there were neither tokens missing nor remaining $(m = 0, r = 0)$, this trace perfectly fits the model *M1*. Similarly, the second and third trace can also be replayed without any problems, i.e., neither tokens are missing nor remaining $(m_2 = m_3 = r_2 = r_3 = 0)$.

Now consider Figure 6, which depicts the replay of the fourth trace from event log *L2* in *M1*. At the beginning (a)(b) the procedure is very similar, only that—instead of transition B—transition C is fired (c), consuming the token from *c1* and producing one token each for *c2* and *c6* $(c = 2, p = 4)$. But when we try to replay the next log event, the corresponding transition H is not enabled. Consequently, the token in *c7* is artificially created and recorded as *missing* $(m = 1)$. Then, transition H is fired (d), consuming the token, and producing one token for place *c8* $(c = 3, p = 5)$. The following log events can be successfully replayed again, i.e., their associated transitions are enabled and can be fired: (e) transition D consuming the token from *c2* and producing one token for *c3* $(c = 4, p = 6)$, (f) transition F consuming the token from *c8* and *c3* and producing one token for *c4* $(c = 6, p = 7)$, (g) one of the two associated transitions A is enabled and can be fired, consuming the token from *c4* and producing one token for the *End* place $(c = 7, p = 8)$. At last, the token at the *End* place is consumed again $(c = 8)$. But then there is still a token *remaining* in place *c6*, which will be punished as it indicates that the process did not complete properly $(r = 1)$. A similar problem will be encountered during the replay of the last trace of event log *L2* (i.e., $r_5 = 1$, $m_5 = 1$).

Using the metric f we can now calculate the fitness between the whole event log *L2* and the process description *M1*. As stated before, besides trace $i = 4$ there were only tokens missing or remaining in the last log trace $i = 5$. Counting also the number of tokens that are produced and consumed while the other three traces are replayed (i.e., $c_2 = c_3 = p_2 = p_3 = 9$, and $c_5 = p_5 = 8$), and with the given number of process instances per trace, the fitness can be measured as $f(M1, L2) = \frac{1}{2}(1 - \frac{51}{10666}) + \frac{1}{2}(1 - \frac{51}{10666}) \approx 0.995$. Similarly, we can calculate the fitness between the event logs *L1*, *L3*, and the process description *M1*, respectively. The first event log *L1* contains only the three log traces that were fitting for *L2*. Thus, there are neither tokens left nor missing in the model during log replay and the fitness measurement yields $f(M1, L1) = 1$. In contrast, for the last event log *L3* none of the traces can be associated with a valid firing sequence of the Petri net, and about half of the produced and consumed tokens were missing or remaining, which leads to a fitness measurement of $f(M1, L3) \approx 0.540$.

As pointed out in Section 3, it is also important to localize a mismatch more closely to give useful feedback to the analyst. In fact, the place of missing and remaining tokens during log replay can provide insight into fitness problems. Consider for example Figure 7, which visualizes some diagnostic information obtained for event log *L2*. Because of the remaining tokens (indicated by a $+$ sign) in place *c6*, transition G remained enabled, and as there were tokens missing (indicated by a $-$ sign) in place *c7*, transition H occurred while this was not possible according to the model. As already discussed, a final interpretation of this mismatch could only be given by a domain expert from the insurance company. However, on a first glance it seems likely that the model lacks the possibility to skip activity G. This is supported by the diagnostics shown in Figure 7.

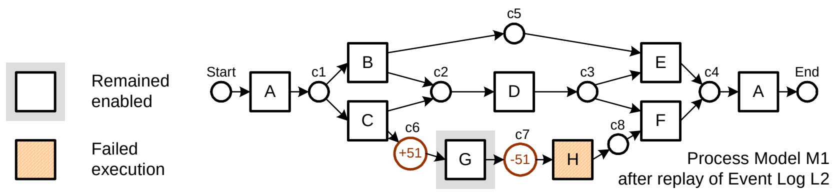


Fig. 7. Diagnostic token counters provide insight into the location of errors: the "+51" in place *c6* indicates that 51 tokens remained ($r = 51$), and the "-51" indicates that 51 times H occurred while it was not enabled ($m = 51$)

Note that this replay is carried out in a non-blocking way and from a log-based perspective, i.e., for each log event in the trace the corresponding transition is fired, regardless of whether the current path of the model is followed or not. This leads to the fact that—in contrast to directly comparing the event streams of models and logs—a series of "missing" log events is punished by the fitness metric f just as much as a single one, since this could always be interpreted as a missing link in the model.

Duplicate tasks cause no problems during log replay as long as exactly one of them is enabled at the same time (like shown in Figure 5 and Figure 6 for the two tasks labeled as A), but otherwise one must enable and/or fire the "right" task for progressing properly. Invisible tasks are considered to be lazy [7], i.e., they are only fired if they can enable the transition in question. In both cases it is necessary to partially explore the state space, which is described in more detail in Section 7.2.

5 Measuring Appropriateness

One way to remove the mismatch visualized in Figure 7 would be to adapt the process model to the process as it really happens (based on the observed behavior in the log), and to introduce an invisible task that enables the skipping of activity *G*. Figure 8 depicts this modified process model *M4*, which is now 100% compliant with event log *L2*.

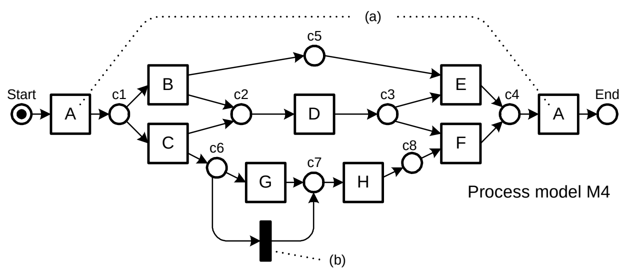


Fig. 8. Model which is 100% compliant with event log *L2*, and which is also appropriate in structure and behavior

But as we have seen in Section 3, the models *M2* and *M3* in Figure 4 are also 100% compliant with event log *L2*, although they do not seem to be “good” models with respect to this log. *M4* in Figure 8 appears to be more *appropriate* from a behavioral perspective (it does not allow for extra behavior as opposed to *M2*), and from a structural perspective (it is more compact and clearly reflects the behavior observed in event log *L2* instead of only listing possible sequences as in *M3*).

In the remainder of this section, both the *behavioral appropriateness* (Section 5.1) and the *structural appropriateness* (Section 5.2) are considered in more detail.

5.1 Behavioral Appropriateness

While fitness evaluates whether every trace in the log is a *possible* execution sequence with respect to the process model, behavioral appropriateness evaluates how much behavior is allowed by the model which was actually *never used* in the observed process executions in the log. The idea is that it is desirable to model a process as precisely as possible. When the model becomes too general and allows for more behavior than necessary (like in the “flower” model *M2*), then it becomes less informative as it no longer describes the actual process, and it may allow for unwanted execution sequences. Consider the process model *M5* in Figure 9. This model is also compliant with event log *L2* as activity *H* can be executed without the previous execution of activity *G*. But, in addition, it allows for arbitrary repetitions of activity *G* (*G* corresponds to the “Consult

Expert" activity in the initial process model in Figure 2), which might not be intended. In this simple example this is rather obvious, but in more complex process models such a problem can be difficult to detect.

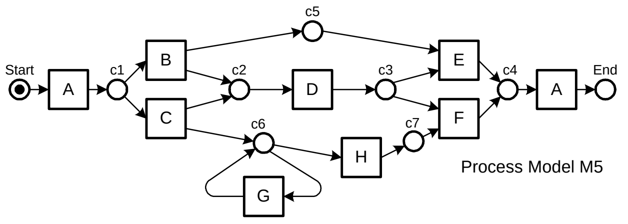


Fig. 9. Model which allows to skip activity G but, in addition, allows for arbitrary repetitions of activity G

Note again that in a practical setting, such a perceived conformance problem can be viewed from two angles. Firstly, the "extra" behavior allowed by the model may correspond to, for example, an alternative branch that deals with an exceptional situation that did not occur within the time frame in which the log was recorded. So in this case, the event log is not complete (cf. Section 2). Secondly, the model may be indeed too generic and allow for situations that never happen in reality. A domain expert will have to be able to differentiate between these two situations. Therefore, suitable metrics are needed.

A first approach to measure the amount of possible behavior is to determine the mean number of enabled transitions during log replay. This corresponds to the idea that an increase of alternatives or parallelism, and therefore an increase of potential behavior, will result in a higher number of enabled transitions during log replay.

Metric 4 (Simple Behavioral Appropriateness) *Let k be the number of* different *traces from the aggregated log. For each log trace i ($1 \leq i \leq k$), n_i is the number of process instances combined into the current trace, and x_i is the mean number of enabled transitions during log replay of the current trace (note that invisible tasks may enable succeeding labeled tasks but they are not counted themselves). Furthermore, T_V is the set of visible tasks in the Petri net model. The simple behavioral appropriateness metric a_B is defined as follows:*

$$a_B = \frac{\sum_{i=1}^{k} n_i(|T_V| - x_i)}{(|T_V| - 1) \cdot \sum_{i=1}^{k} n_i}$$

Assuming that $|T_V| > 1$, this metric ranges from 0 (if all visible tasks in the model are always enabled during log replay, such as it is the case in the "flower" model *M2*) to 1 (a sequential process)[3]. If we calculate the simple behavioral

[3] Note that we here assume 100% fitness and, therefore, there is always at least one transition enabled during log replay.

appropriateness for *M4*, the metric yields $a_B(M4, L2) \approx 0.967$. This is a slightly bigger value than for the model that allows for arbitrary loops of activity G (*M5*), which yields $a_B(M5, L2) \approx 0.964$.

However, there is the problem that this metric can only be used as a comparative means, because it measures the appropriateness relatively to the degree of *model flexibility*. That is, model *M4* is better than model *M5*, because the less behavior is allowed by the model the better. But it only reaches the value 1 in a purely sequential model, where exactly one task is enabled in each step of the log replay. In addition, the metric is not stable to situations where the model is sequentialized through duplicate tasks, such as process model *M3* in Figure 4(b).

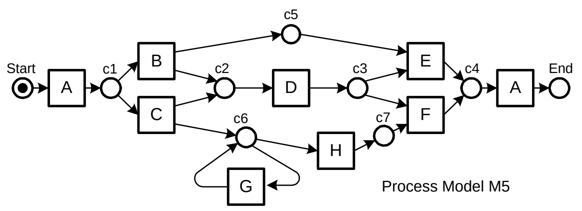
To approach behavioral appropriateness independently from such structural properties, and independently of the model flexibility, the potential behavior specified by the model must be analyzed and compared with the behavior *actually needed* to describe what was observed in the log. The notion of a set of labels that serves as a link between the tasks in the model and the elements contained in the log (cf. Section 2) makes it possible to derive comparable "Follows" and "Precedes" relations between activities from both a model and a log perspective. To weaken the completeness requirement towards the event log, and to also capture long-distance dependencies between activities, the "Follows" (or "Precedes") relation is determined *globally* (i.e., the tasks, or log events, do not need to *directly* follow or precede each other). If we then look at a *set of sequences*, we can determine whether two activities (x, y) either *always*, *never*, or *sometimes* follow or precede each other:

Definition 1 (Follows relations) *Two activities* (x, y) *are in "Always Follows", "Never Follows", or "Sometimes Follows" relation in the case that, if* x *is executed at least once, then* always*,* never*, or* sometimes *also* y *is eventually executed, respectively.*

Definition 2 (Precedes relations) *Two activities* (x, y) *are in "Always Precedes", "Never Precedes", or "Sometimes Precedes" relation in the case that, if* y *is executed at least once, then* always*,* never*, or* sometimes *also* x *was executed some time before, respectively.*

Note that the "Follows" and the "Precedes" relations are defined as soon as they hold for *any* pair of labels in a sequence. To give an example, imagine a sequence $(x, ..., x, ..., y, ..., x)$. Here, the tuple (x, y) would be an element of the "Follows" relation, although it does not hold for all x that they are eventually followed by y.

Consider Figure 10, which illustrates the global "Follows" relations that are derived from model *M5* and event log *L2*. To build these relations from a model perspective, we analyze the possible execution sequences (based on a state space analysis or "exhaustive" simulation of the model). From a log perspective we analyze the observed execution sequences ("walking through" the log). From this, we can determine whether two activities (x, y) either *always*, *never*, or



No. of Instances	Log Traces
1207	ABDEA
145	ACDGHFA
56	ACGDHFA
23	ACHDFA
28	ACDHFA

Event Log L2

Analyze whether activities in the model Always (A), Never (N), or Sometimes (S) follow each other

Analyze whether events in the log actually Always (A), Never (N), or Sometimes (S) followed each other

F	A	B	C	D	E	F	G	H
A	A	S	S	A	S	S	S	S
B	A	N	N	A	A	N	N	N
C	A	N	N	A	N	A	S	A
D	A	N	N	N	S	S	S	S
E	A	N	N	N	N	N	N	N
F	A	N	N	N	N	N	N	N
G	A	N	N	S	N	A	S	A
H	A	N	N	S	N	A	N	N

(a) "Follows" relations from model perspective

F	A	B	C	D	E	F	G	H
A	A	S	S	A	S	S	S	S
B	A	N	N	A	A	N	N	N
C	A	N	N	A	N	A	S	A
D	A	N	N	N	S	S	S	S
E	A	N	N	N	N	N	N	N
F	A	N	N	N	N	N	N	N
G	A	N	N	S	N	A	N	A
H	A	N	N	S	N	A	N	N

(b) "Follows" relations from log perspective

Fig. 10. Global "Follows" relations derived for model *M5* and event log *L2*

sometimes follow each other. The same can be done for the global "Precedes" relations. In Figure 10 one can see that while according to the model *M5* activity G may be followed by activity G (i.e., (G, G) is an element of the "Sometimes Follows" relation), this actually never happened in event log *L2* (i.e., (G, G) is an element of the "Never Follows" relation). We refer to a technical report [31] for a detailed and formal description of these relations. Note that in general the number of paths in the model is larger than the set of traces actually appearing in the log. Therefore, the cost of deriving the relations from the model may be problematic, while constructing them from the log is typically no problem (cf. Section 7.2 for some complexity indications).

While the "Always" and "Never" relations describe *hard constraints* (i.e., "Follows" or "Precedes" relations that always or never hold for a sequence of activities), the "Sometimes" relations capture *variabilities* in behavior. For example, concurrent activities may follow and precede each other in any order (cf. (D, H) and (H, D) in Figure 10). Similarly, activities preceding a number of alternative branches are sometimes followed by one of these alternative branches and sometimes by another (cf. (A, B) and (A, C) in Figure 10). The same holds for activities that follow after a number of alternative branches were joined (reflected in the "Sometimes Precedes" relations). Therefore, the idea of the following metric is to compare the variabilities of the behavior allowed by the model and the behavior observed in the log based on the cardinal numbers of the S_F and S_P relations.

Metric 5 (Advanced Behavioral Appropriateness) *Let S_F^m be the S_F relation and S_P^m be the S_P relation for the process model, and S_F^l the S_F relation and S_P^l the S_P relation for the event log. The advanced behavioral appropriateness metric a'_B is defined as follows:*

$$a'_B = \left(\frac{|S_F^l \cap S_F^m|}{2 \cdot |S_F^m|} + \frac{|S_P^l \cap S_P^m|}{2 \cdot |S_P^m|}\right)$$

Note that—for a rather technical reason—the set of labels, which are considered to form these relations, includes an artificially inserted *Start* and *End* task or log event, respectively, which is abstracted from in this paper. Note further that we build the intersection of S_F^l and S_F^m (and S_P^l and S_P^m) to look only where the log becomes more specific, i.e., we capture situations where—according to the model—two activities may *sometimes* follow each other (and sometimes not), but in the log they *always* or *never* follow each other. The reverse can also happen, i.e., the model is more specific than the log, which then indicates a fitness problem. However, since we discard these tuples, the values assigned by a'_B range from 0 to 1. Note finally that although the S_F and S_P relations are symmetric, we consider both and weigh them equally to make the metric stable with respect to the position of the "extra behavior".

If we calculate this new behavioral appropriateness metric for model *M4*, the metric yields $a'_B(M4, L2) = 1$, which indicates that the model *M4* precisely allows for the behavior that was observed in event log *L2*. For process model *M5* it yields $a'_B(M5, L2) = (\frac{19}{2 \cdot 20} + \frac{20}{2 \cdot 21}) \approx 0.951$. The sometimes relations that are derived from the model contain one element more than the sometimes relations that are derived from the log, which is the element (G, G) (according to the model activity G may be followed or preceded by itself). Finally, calculating the value for the model *M2* yields $a'_B(M2, L2) \approx 0.271$. Note that, because for the new metric the actual distance between model and log relations is considered, the "flower" model can also be a "good" model, namely if the event log itself exhibits random behavior.

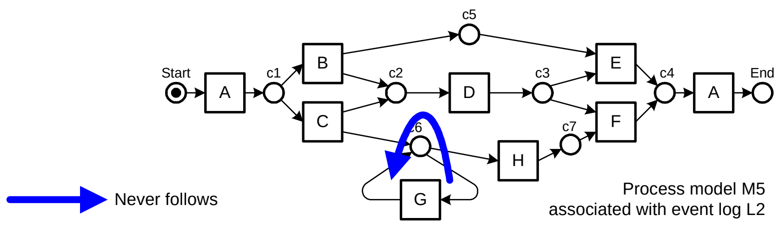


Fig. 11. Differences in successor and predecessor relationships can be visualized

Building on a notion of global successor and predecessor relationships, we are also able to highlight "unused" alternative and concurrent parts in the model, which can be visualized, for example, as indicated in Figure 11.

5.2 Structural Appropriateness

The desire to model a business process in a compact and meaningful way is difficult to capture by measurement. Whether a model is perceived as suitable may depend on subjective preferences, and is typically correlated to the specific purpose of the model. There are aspects like, for example, the granularity of the described workflow actions, which can only be determined by an experienced human designer. But the notion of structural appropriateness addressed by this paper rather relates to the control flow perspective, and often there are *several syntactic ways to express the same behavior* in a process model. Consider, for example, model *M6* in Figure 12, which allows for the same behavior[4] as model *M4* and model *M3*. However, it contains the following constructs that may "inflate" the structure of a process model, and therefore render it less compact and understandable.

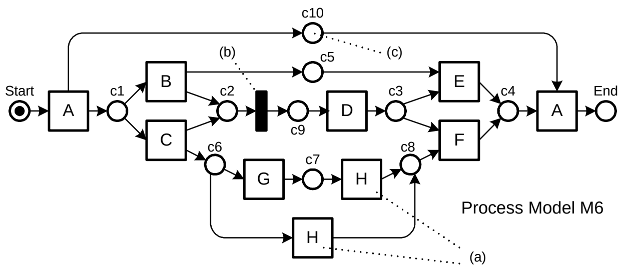


Fig. 12. Model containing some constructs that may "inflate" the structure of a process model: (a) duplicate tasks, (b) invisible task, (c) implicit place

(a) *Duplicate tasks.* In addition to duplicate tasks that are necessary to specify that a certain activity takes place in a completely different context, such as at the beginning and at the end of the process like task A in process model *M4* (see (a) in Figure 8), there are also duplicate tasks that could be "folded" as the different contexts of their execution can be captured in the model. For example, in model *M6* a duplication of task H is used to express that after performing activity C either the sequence GH or H alone can be executed (see (a) in Figure 12). Figure 8 (process model *M4*) describes the same process with the help of an invisible task (see (b) in Figure 8). Duplicate tasks can reduce the structural

[4] Note that there exist many equivalence notions for process models. Here, we assume *trace equivalence*: two models are considered equivalent if the sets of traces they can execute are identical.

appropriateness of a model because they prevent abstraction (it cannot be easily seen anymore from the model that two tasks are actually the same). The model *M3* shows the extreme case of a completely instance-based view on the process with many superfluous duplicate tasks.

(b) *Invisible tasks.* Besides the invisible tasks used for routing purposes like, e.g., indicated in Figure 8(b), there are also invisible tasks that only delay visible tasks, such as the one indicated by (b) in Figure 12. If they do not serve any other purpose they can simply be removed, thus making the model more concise.

(c) *Implicit places.* Implicit places are places that can be removed without changing the behavior of the model [11]. An example for an implicit place is given by place *c10* (see (c) in Figure 12). Note that the place *c5* in Figure 12 is not implicit as it influences the choice made later on between E and F. Both *c5* and *c10* are *silent places*, with a silent place being a place whose directly preceding transitions are never directly followed by one of their directly succeeding transitions (e.g., for *M4* it is not possible to produce an event sequence containing BE or AA). Process discovery techniques by definition are unable to detect implicit places, and have problems detecting silent places.

Note that these constructs are only an indicator for a potential conformance problem. For example, there may well be situations in which a modeler finds it more convenient to model a situation using duplicate tasks although it could be avoided (because this, for example, eliminates potential synchronization tasks that would be otherwise needed). Moreover, it may be useful to explicitly denote a certain partial state (such as "machine busy") with an implicit place. However, the detection of such potentially problematic constructs can help the business analyst to systematically assess the process model at hand.

As a first indicator for structural appropriateness we define a simple metric based on the number of different task labels in relation to the graph size of the model.

Metric 6 (Simple Structural Appropriateness) *Let L be the set of labels that establish the mapping between tasks in the model and events in the log, and N the set of nodes (i.e., places and transitions) in the Petri net model. The simple structural appropriateness metric a_S is defined as follows:*

$$a_S = \frac{|L| + 2}{|N|}$$

Given the fact that a WF-net (cf. Section 2) is expected to have a dedicated *Start* and *End* place, the graph must contain at least one transition for every task label, plus two places (the start and end place). In this case $|N| = |L| + 2$ and the metric a_S yields the value 1. The more the size of the graph is growing, e.g., due to additional places, the measured value moves towards 0.

If we calculate the structural appropriateness for the model *M3*, it yields $a_S(M3) \approx 0.170$, which is a very bad value caused by the many duplicate tasks (as they increase the number of transitions while having identical labels). For the model *M4* the metric yields $a_S(M4) = 0.5$. A slightly lower value $a_S(M6) \approx 0.435$ is calculated for the model in Figure 12.

However, this metric can only be used as a comparative means for process models that exhibit equivalent behavior (because it is only based on the graph size of the model). Therefore, it is of limited applicability.

To approach structural appropriateness independently of the actual behavior of the model, it is a better idea to verify certain design guidelines, which define the *preferred* way to express specific behavioral patterns, and to somehow punish violations of these guidelines. It is obvious that the design guidelines will vary for different process modeling notations and may depend on personal or corporate preferences. Nevertheless, in the following we present a new structural appropriateness approach based on the findings reported earlier in this section. As a design guideline, constructs such as *alternative duplicate tasks* (duplicate tasks that never happen together in one execution sequence) and *redundant invisible tasks* (invisible tasks that can be removed from the model without changing the behavior) should be avoided as they were identified to inflate the structure of a process model and to detract from clarity in which the expressed behavior is reflected. A more complete description including a formal specification of the approach can be found in a technical report [31]. Note that because the number of paths in the model can become very large, the cost of detecting alternative duplicate tasks may be problematic. In contrast, redundant invisible tasks can be detected via structural analysis of the model, which is typically very fast (cf. Section 7.2 for some complexity indications).

Metric 7 (Advanced Structural Appropriateness) *Let T be the set of transitions in the Petri net model, T_{DA} the set of alternative duplicate tasks, T_{IR} the set of redundant invisible tasks. The advanced structural appropriateness metric a'_S is defined as follows:*

$$a'_S = \frac{|T| - (|T_{DA}| + |T_{IR}|)}{|T|}$$

Note that $|T_{DA}| + |T_{IR}| \leq |T|$ and therefore $0 \leq a'_S \leq 1$ as duplicate tasks are always visible. Revisiting the example models it becomes clear that—according to the defined design guideline—only model *M6* and *M3* are reduced in structural appropriateness. For *M6* the number of alternative duplicate tasks $|T_{DA}| = 2$ (see (a) in Figure 12) and the number of redundant invisible tasks $|T_{IR}| = 1$ (see (b) in Figure 12), which results in $a'_S(M5) \approx 0.727$. In *M3* all tasks but A, B and E belong to the set of alternative duplicate tasks and therefore $a'_S(M3) \approx 0.387$.

Building on some kind of design guideline, we are usually also able to locate its violations and visualize them such as, for example, indicated in Figure 13.

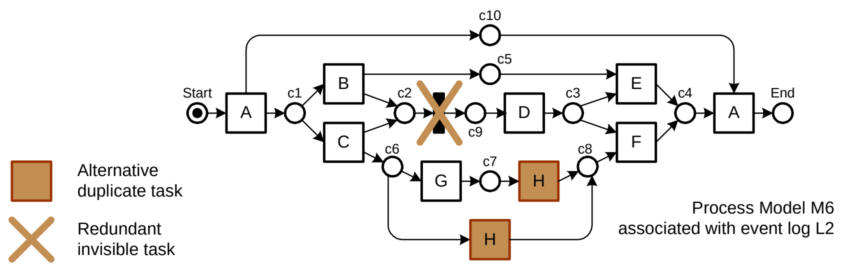

Start
A
c1
B
C
c2
c9
D
c3
c5
c10
E
F
c4
A
End
c6
G
c7
H
c8
H
Alternative duplicate task
Redundant invisible task
Process Model M6 associated with event log L2

Fig. 13. Design guideline violations can be visualized

6 Balancing Fitness and Appropriateness

In general, the presented notions of conformance, i.e., fitness, behavioral appropriateness, and structural appropriateness are orthogonal to each other. They measure something completely different and, therefore, an improvement according to one notion is not really comparable to an improvement according to another notion.

We can use the defined conformance metrics to position the example models and logs with respect to fitness, behavioral appropriateness, and structural appropriateness. Table 1 contains the measured values for all combinations of example models and logs in this paper. If we equally weigh the three metrics f, a'_B, and a'_S, then process model *M1* is the overall best conforming model for event log *L1*, *M4* has the best conformance with respect to event log *L2*, and *M2* with respect to event log *L3*.

Table 1. Overview of the values of the defined conformance metrics for all combinations of example models and logs in this paper

	M1	*M2*	*M3*	*M4*	*M5*	*M6*
	$f = 1.0$	$f = 1.0$	$f = 1.0$	$f = 1.0$	$f = 1.0$	$f = 1.0$
	$a_B = 0.9740$	$a_B = 0.0$	$a_B = 0.9739$	$a_B = 0.9718$	$a_B = 0.9703$	$a_B = 0.9749$
L1	$a'_B = 0.9167$	$a'_B = 0.2292$	$a'_B = 0.8474$	$a'_B = 0.8474$	$a'_B = 0.8060$	$a'_B = 0.8474$
	$a_S = 0.5263$	$a_S = 0.7692$	$a_S = 0.1695$	$a_S = 0.5$	$a_S = 0.5556$	$a_S = 0.4348$
	$a'_S = 1.0$	$a'_S = 1.0$	$a'_S = 0.3871$	$a'_S = 1.0$	$a'_S = 1.0$	$a'_S = 0.7273$
	$f = 0.9952$	$f = 1.0$	$f = 1.0$	$f = 1.0$	$f = 1.0$	$f = 1.0$
	$a_B = 0.9705$	$a_B = 0.0$	$a_B = 0.9745$	$a_B = 0.9669$	$a_B = 0.9637$	$a_B = 0.9706$
L2	$a'_B = 1.0$	$a'_B = 0.2708$	$a'_B = 1.0$	$a'_B = 1.0$	$a'_B = 0.9512$	$a'_B = 1.0$
	$a_S = 0.5263$	$a_S = 0.7692$	$a_S = 0.1695$	$a_S = 0.5$	$a_S = 0.5556$	$a_S = 0.4348$
	$a'_S = 1.0$	$a'_S = 1.0$	$a'_S = 0.3871$	$a'_S = 1.0$	$a'_S = 1.0$	$a'_S = 0.7273$
	$f = 0.5397$	$f = 1.0$	$f = 0.4947$	$f = 0.6003$	$f = 0.5830$	$f = 0.6119$
	$a_B = 0.8909$	$a_B = 0.0$	$a_B = 0.8798$	$a_B = 0.8904$	$a_B = 0.8894$	$a_B = 0.9026$
L3	$a'_B = 0.75$	$a'_B = 0.4583$	$a'_B = 0.7434$	$a'_B = 0.7434$	$a'_B = 0.7071$	$a'_B = 0.7434$
	$a_S = 0.5263$	$a_S = 0.7692$	$a_S = 0.1695$	$a_S = 0.5$	$a_S = 0.5556$	$a_S = 0.4348$
	$a'_S = 1.0$	$a'_S = 1.0$	$a'_S = 0.3871$	$a'_S = 1.0$	$a'_S = 1.0$	$a'_S = 0.7273$

Note that the metrics a_B and a_S are not considered because it was shown that they are not *stable* enough (cf. Section 3) to compare all process models with each other. Recall that, for example, the metric a_B is affected by structural properties. Nevertheless, the metrics a_B and a_S can be well applied as a comparative means within the given restrictions. So, for example, the a_B metric determines *M1*, the initial Petri net given in Figure 2, as the behaviorally most suitable model over *M2*, *M4*, and *M5* with respect to event log *L1*.

Although, ideally, a process model and a log should have both 100% fitness, and behavioral and structural appropriateness, it can be expected that in a practical setting the fitness dimension is typically more dominant. Therefore, we recommend to carry out the conformance analysis in two phases (first, the fitness is analyzed, and then the appropriateness of the model is assessed afterwards).

7 Adding Conformance to the ProM Framework

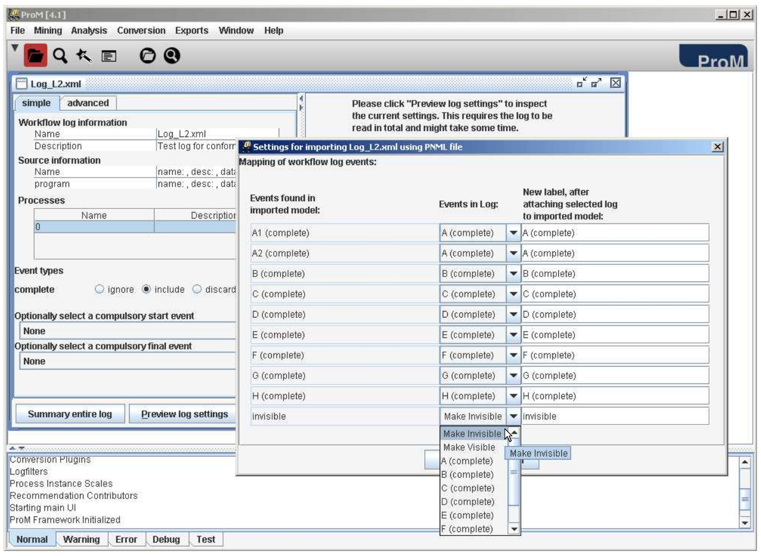
The Process Mining (ProM) framework is an extensible tool suite that supports a wide variety of process mining techniques in the form of plug-ins [3]. In this section, we describe how the concepts presented in this paper are supported by the ProM tool. For this, we first give an overview about the provided functionality in Section 7.1, and then highlight some challenges related to the log replay involving invisible and duplicate tasks in Section 7.2.

7.1 Functionality of the Conformance Analysis Plug-in

The Conformance Checker[5] replays an event log within a Petri net model in a non-blocking way while gathering diagnostic information that can be accessed afterwards. It calculates the token-based fitness metric f (taking the number of process instances for each log trace into account), the behavioral appropriateness metrics a_B and a'_B, and the structural appropriateness metrics a_S and a'_S.

During log replay the plug-in takes care of invisible tasks that might enable the transition to be replayed next, and it is able to deal with duplicate tasks (see also Section 7.2). Figure 14 shows a screenshot of the ProM framework establishing the mapping between process model *M4* and event log *L2*. While the left column lists all the different transitions contained in the Petri net model, each of them can either be related to a log event contained in the associated log, or made invisible. A third possibility is to make it visible without linking it to an event in the log, which is needed if this activity never occurred in the log (cf. Section 2.3). In the situation shown in Figure 14 the tasks *B (complete)* to *H (complete)* are all one-to-one mapped onto different log events, while *A1 (complete)* and *A2 (complete)* are both related to the same log event *A (complete)*, i.e., they are duplicate tasks. Moreover, the task with the name *invisible* is made invisible. Note that for practical use the mapping has been made explicit, so any

[5] Both the Conformance Checker, which is embedded in the ProM framework, and the files belonging to the example logs and models used in this paper can be downloaded from *http://www.processmining.org*.


ProM [4.1]
File Mining Analysis Conversion Exports Window Help
ProM
Log_L2.xml
simple advanced
Summary entire log
Preview log settings
Normal Warning Error Debug Test

Workflow log information

Name	Log_L2.xml
Description	Test log for confor

Source information

Name	name: , desc: , dat
program	name: , desc: , dat

Processes

Name	Descriptio
0	

Event types

complete ○ ignore [● include] ○ discard

Optionally select a compulsory start event: None

Optionally select a compulsory final event: None

Please click "Preview log settings" to inspect the current settings. This requires the log to be read in total and might take some time.

Settings for importing Log_L2.xml using PNML file

Mapping of workflow log events:

Events found in imported model:	Events in Log:	New label, after attaching selected log to imported model:
A1 (complete)	A (complete)	A (complete)
A2 (complete)	A (complete)	A (complete)
B (complete)	B (complete)	B (complete)
C (complete)	C (complete)	C (complete)
D (complete)	D (complete)	D (complete)
E (complete)	E (complete)	E (complete)
F (complete)	F (complete)	F (complete)
G (complete)	G (complete)	G (complete)
H (complete)	H (complete)	H (complete)
invisible	Make Invisible	invisible

Make Invisible
Make Visible
A (complete)
B (complete)
C (complete)
D (complete)
E (complete)
F (complete)

Conversion Plugins
Logfilters
Process Instance Scales
Recommendation Contributors
Starting main UI
ProM Framework Initialized

Fig. 14. Screenshot of ProM while associating model tasks with log events

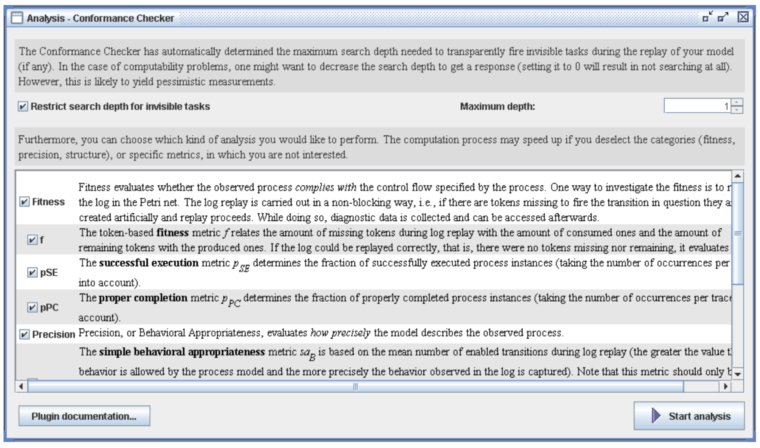
task label can be set in the right column (the name of the task, the name of the log event, or even something else). All audit trail entries that are not mapped to a task in the model are automatically removed from the log.

Then, the Conformance Checker can be started. The settings screen shown in Figure 15 will appear first, and the user can select which metrics should be calculated. The Conformance Checker supports all the metrics presented earlier in this paper, and also provides visualizations for detected conformance problems. Furthermore, it automatically determines the maximum length of possible sequences of invisible tasks in the model for an efficient log replay (cf. Section 7.2). Later, in connection with the case studies, we will show screenshots of the plug-in. However, first we elaborate on the implementation.

7.2 Implementation of the Conformance Analysis Plug-in

To calculate the presented metrics the Conformance Checker makes use of the following analysis methods:

- *State space analysis*, i.e., the coverability graph [16, 17, 27, 29] of the process model is traversed while loops are followed at most twice. This is used for the calculation of both the metrics a'_B (deriving the activity relations from the model perspective) and a'_S (detecting alternative duplicate tasks).
  Because the state space of a model can grow exponentially, state-based analysis techniques may be problematic with respect to computational complexity. However, there exist techniques for state space reduction, such as partial order reduction, and symmetry methods, which may be exploited. Moreover,



The Conformance Checker has automatically determined the maximum search depth needed to transparently fire invisible tasks during the replay of your model (if any). In the case of computability problems, one might want to decrease the search depth to get a response (setting it to 0 will result in not searching at all). However, this is likely to yield pessimistic measurements.

✓ Restrict search depth for invisible tasks — Maximum depth: 1

Furthermore, you can choose which kind of analysis you would like to perform. The computation process may speed up if you deselect the categories (fitness, precision, structure), or specific metrics, in which you are not interested.

✓ Fitness	Fitness evaluates whether the observed process *complies with* the control flow specified by the process. One way to investigate the fitness is to r the log in the Petri net. The log replay is carried out in a non-blocking way, i.e., if there are tokens missing to fire the transition in question they a created artificially and replay proceeds. While doing so, diagnostic data is collected and can be accessed afterwards.
✓ f	The token-based **fitness** metric *f* relates the amount of missing tokens during log replay with the amount of consumed ones and the amount of remaining tokens with the produced ones. If the log could be replayed correctly, that is, there were no tokens missing nor remaining, it evaluates
✓ pSE	The **successful execution** metric p_{SE} determines the fraction of successfully executed process instances (taking the number of occurrences per into account).
✓ pPC	The **proper completion** metric p_{PC} determines the fraction of properly completed process instances (taking the number of occurrences per trac account).
✓ Precision	Precision, or Behavioral Appropriateness, evaluates *how precisely* the model describes the observed process.
	The **simple behavioral appropriateness** metric sa_B is based on the mean number of enabled transitions during log replay (the greater the value t behavior is allowed by the process model and the more precisely the behavior observed in the log is captured). Note that this metric should only b

Fig. 15. Screenshot of the conformance analysis settings

e.g., the activity relations for metric a'_B serve as a footprint and may be approximated. For example, one could stop constructing the state space after some time or space limit is reached and construct the footprint based on this. Hence, it is possible to balance efficiency and precision.

– *Structural analysis*, i.e., the structure of the process model is analyzed. This is used for the metrics a_S (assessing the graph size) and a'_S (detecting redundant invisible tasks). Note that the redundant invisible tasks are distinguished via reduction rules similar to [4] and based on [27].
  Compared to the state space analysis, structural analysis techniques are typically very efficient.
– *Log replay analysis*, i.e., the log is replayed in a non-blocking way and from a log perspective. This is necessary for calculating the metrics f (measuring the amount of consumed, produced, missing and remaining tokens) and a_B (measuring the mean number of enabled transitions). To derive the activity relations from a log perspective for metric a'_B a single pass of the log is sufficient (i.e., no actual log replay is needed).
  The time complexity of the log replay method without invisible or duplicate tasks increases only linearly with the size of the log. This is very important for practical applicability as it also enables the analysis of large logs. However, if the log replay involves invisible or duplicate tasks, there may be situations in the course of replay where the state space of the process model needs to be partly explored, which may degrade the performance.

In the remainder of this section we concentrate on the log replay method and show how we approach two non-trivial problems, namely whether a specific task can be enabled via firing a sequence of invisible tasks (see Algorithm 1) and the decision for one task among duplicates (see Algorithm 2). Note that the presented algorithms are heuristic, local approaches that deal with the replay of the next step in the log trace. Unfortunately, this means that it cannot be guaranteed that if the log fits the model it can be replayed correctly (and thus any mismatch really indicates a conformance problem). For example, we choose the shortest sequence of invisible tasks to enable the currently replayed task if possible. However, from a global viewpoint it could always be the case that firing some longer sequence would actually produce exactly those tokens that are needed in a later stage of the replay. Dealing with this issue in a global manner (i.e., minimizing the number of missing and remaining tokens during log replay) seems intractable for complexity reasons[6]. However, the presented algorithms work well in most of the cases, while keeping the technique accessible for practical situations. The two algorithms are described in the remainder of this section.

The first algorithm deals with the fact that invisible tasks are considered to be lazy, i.e., they might fire to enable one of their succeeding visible tasks, but will never be fired directly in the course of log replay since they do not have a log event associated. This implies that in the case that the task currently replayed is not directly enabled, it must be checked whether it can be enabled by a sequence of invisible tasks before considering it having failed. If there are multiple enabling sequences, we choose the shortest sequence among them (or one of the shortest sequences if there are more than one that are "the shortest"). This heuristic aims at having minimal possible side effects on the current marking of the net, e.g., not to unnecessarily fire an invisible task that is in conflict with another task later to be replayed. As indicated earlier, there may be situations where a longer sequence of invisible tasks results in a better replay than a shorter sequence. However, while an optimal solution would be difficult from a complexity point of view, our "best effort" solution seems to work well in practice. Algorithm 1 shows the flow of the method `isEnabled()`.

First, a `list` is created to capture the (potential) enabling sequence. If the transition is already enabled, this sequence remains empty and the method returns `true`. Note that an empty list has length 0 but is not `NIL` (which can be seen as *undefined*). In the case the transition is not directly enabled, the state space[7] is built from the current marking of the Petri net. To prevent nets that accumulate tokens from producing an infinite state space (which could happen, e.g., when building the reachability graph of a Petri net) the coverability graph builder of the ProM framework has been used for implementation. In a coverability graph a so-called ω-state denotes an extended marking, which subsumes all the different finite markings that result from token accumulation in an infinite marking [27, 35]. Then `traceShortestPathOfInvisibleTasks()`—the recursive program part—is called. The idea is to look for a sequence of invisible tasks that can be fired to

[6] Note that the theoretical worst-case complexity of generating a coverability graph is non-primitive recursive space, although for small to medium sized systems (up to 100 transitions) generating a coverability graph is often feasible [35].

[7] Note that in fact we only need to build a *partial* state space depending on the maximum length of possible sequences of invisible tasks in the model. This maximum depth can be efficiently calculated based on the structure of the model, and is automatically determined by the Conformance Checker.

Algorithm 1 Recursive method for transparently enabling a replayed task through a sequence of invisible tasks (if possible)

```
$isEnabled$ :
 1: $list \leftarrow$ new empty list
 2: $soFarShortestPath \leftarrow NIL$
 3: if not directly enabled then
 4:    clone Petri net and build state space from current marking
 5:    $list \leftarrow traceShortestPathOfInvisibleTasks(...)$
 6: end if
 7: if $list = NIL$ then
 8:    return $false$
 9: else
10:    while $list$ has next element do
11:       fetch next task from list
12:       fire corresponding transition in Petri net
13:    end while
14:    return $true$
15: end if

$traceShortestPathOfInvisibleTasks$ :
 1: if current state already visited $\vee$ shorter path already found then
 2:    return $soFarShortestPath$                                        // (a) (b)
 3: else
 4:    while possible path from current state in state space left do
 5:       determine next task
 6:       if requested task found then
 7:          return $currentPath$                                         // (c)
 8:       else if invisible task found then
 9:          set current state visited
10:          copy $currentPath$ and append task
11:          determine next state
12:          $soFarShortestPath \leftarrow traceShortestPathOfInvisibleTasks(...)$
13:       else
14:          return $NIL$                                                 // (d)
15:       end if
16:    end while
17:    return $soFarShortestPath$
18: end if
```

create a marking of the Petri net that allows to execute the currently replayed transition (in this case, the transition is considered to be enabled, and there is no conformance problem). For this, each possible path of invisible tasks in the state space is traced until one of the following end-conditions is reached:

(a) If the current state has been already visited during traversal, it means that the state space is cyclic and recursion stops to prevent an infinite loop.
(b) If a shorter sequence of invisible tasks enabling the transition in question than the one currently traced has already been found, it is not necessary to pursue this route any further.

Assuming that neither (a) nor (b) are fulfilled, the current state in the state space is marked as visited, and all possible paths spawned from this state are considered and further traced until one of the following end conditions holds:

(c) If the transition to be replayed is encountered, a possible enabling sequence has been found and will be returned.
(d) If no invisible task can be found, recursion aborts as the path cannot be followed any further.

If no possible sequence could be found, i.e., the checked transition cannot be enabled via firing any invisible tasks either, the `list` will be set to `NIL` and the method returns `false`. But in the case a possible path *has* been found, the selected sequence of invisible tasks is executed to enable the transition and the method returns `true`.

The second algorithm deals with the fact that the mapping between model tasks and log events may result in duplicate tasks. During log replay this is a problem since for a log event that is associated with multiple tasks in the model, it is not always clear which of the duplicates should be executed. Algorithm 2 shows the flow of the implemented method `chooseEnabledDuplicateTask()`, which is called in the case that more than one transition is found associated with the log event currently replayed.

At first, only those tasks that are enabled[8] by the current marking of the Petri net are selected. If none of them is enabled, the method returns immediately and the Conformance Checker will fire an arbitrary task from the list of duplicates (since correct replay is not possible anyway). If there is exactly one task enabled, it is returned and will be executed subsequently. This will often be the case in scenarios where the same task is carried out in multiple contexts (such as setting a checkpoint in the beginning and in the end of the example process in Figure 2), i.e., the marking of the net clearly indicates which choice is best. However, if there are more candidates enabled, the remaining log events must be considered to determine the best choice. For these purposes, `chooseBestCandidate()` is called. It makes a copy of the current replay scenario for each enabled duplicate and fires the transition belonging to that candidate (i.e., it starts to mimic every

[8] Note that in the context of this algorithm the possibility of invisible tasks indirectly enabling other transitions needs to be respected again (but without actually changing the marking of the replayed net). Nevertheless, from now on we abstract from this.

Algorithm 2 Recursive method for choosing one task among a set of duplicate tasks during log replay

```
chooseEnabledDuplicateTask :
 1: candidateList ← select all enabled duplicates
 2: if length of candidateList = 0 then
 3:    return NIL
 4: else if length of candidateList = 1 then
 5:    return the only enabled duplicate
 6: else
 7:    clone replay scenario for each candidate and fire corresponding transition
 8:    return traceBestCandidate(...)
 9: end if

traceBestCandidate :
 1: if no log events left then
 2:    return any of remaining candidateList                              // (a)
 3: else
 4:    fetch next log event
 5:    nextTask ← determine transition(s) associated to log event
 6:    for each scenario from candidateList do
 7:       if nextTask is duplicate task then
 8:          nextTask ← chooseEnabledDuplicateTask(...)
 9:       else if nextTask is not enabled then
10:          nextTask ← NIL
11:       end if
12:       if nextTask = NIL then
13:          remove current from candidateList
14:       else
15:          further trace replay scenario via firing nextTask
16:       end if
17:    end for
18: end if
19: if length of candidateList = 0 then
20:    return any of remaining candidateList                              // (b)
21: else if length of candidateList = 1 then
22:    return the only remaining candidate                                // (c)
23: else
24:    return traceBestCandidate(...)
25: end if
```

case). Then the entry point for the recursive method tracking these scenarios `traceBestCandidate()` is reached and will not return until there is only one scenario left, which can then be reported to the initial caller to proceed with the actual log replay. First, the following end-condition is checked:

(a) If there are no log events left in the trace currently replayed, then one of the remaining candidates is chosen arbitrarily and recursion finishes.

Assuming that (a) is not fulfilled the next log event is fetched from the trace and the number of associated transitions is determined. If there is only one task associated to it, those scenarios are kept and updated where this task is enabled, i.e., where the next replay step can be executed successfully as well. If there are multiple tasks associated, the best duplicate must also be chosen for this case and for each scenario, realized by a recursive call to the very entry point of the whole procedure. Then, similarly, those scenarios are kept and updated that were able to determine an enabled duplicate task for this anticipated next replay step. The possibility for having a 0:1 mapping has been discarded since log events not associated to any task in the model were removed during import (cf. Section 2.3).

Now, the number of remaining scenarios is checked and if there are more than one left, recursion proceeds to check at least one step further. Otherwise one of the two following end-conditions is reached:

(b) If only a single candidate remains, this one is returned as the best choice.
(c) If after the replay of this next log event none of the scenarios is left, any of the previously kept candidates is returned.

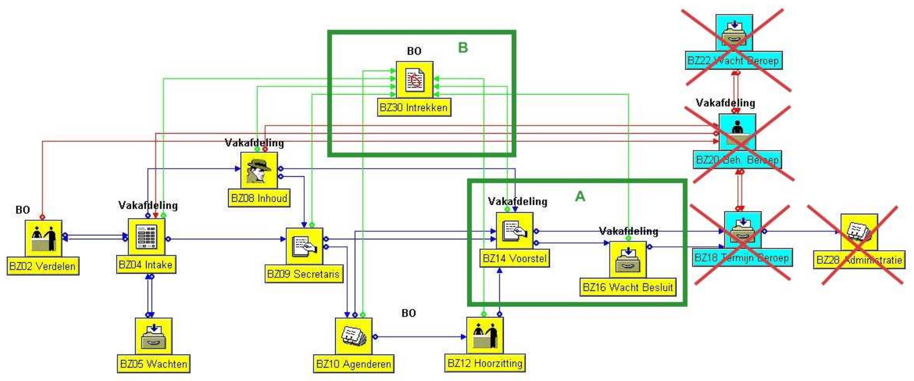
8 Conformance Checking Applications

In the following subsections we describe two different applications of the presented conformance checking techniques. The first application involves administrative processes of a municipality in the Netherlands (Section 8.1), whereas in the second case the Conformance Checker has been used to analyze (web) service behavior (Section 8.2).

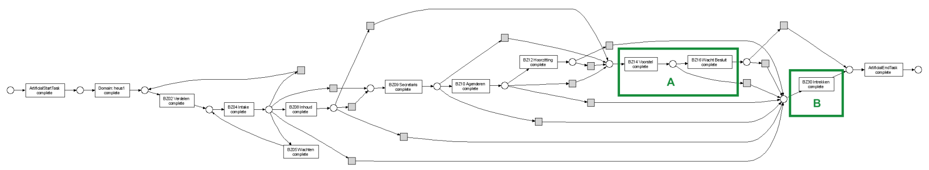
8.1 Town Hall

In the context of a project cooperating with a town hall in the Netherlands we had the opportunity to apply our techniques to real-life logs related to four different administrative processes. Three processes deal with the handling of complaints and the fourth process handles building permit applications. As an example, Figure 16(a) shows the original description of one of the complaint handling procedures. It has been created in a tool called “Routebuilder”, which comes with the Global 360 Enterprise EX workflow system (formerly known as eiStream WMS). Note that all the considered tasks have XOR-split/join semantics.

The managers at the municipality were especially interested in answers to the following questions: “Are there deviations from the designed process?”, “What



(a) The original description of the complaint handling procedure as used by the G360 Enterprise EX workflow system



(b) The Petri net model of the complaint handling procedure loaded into ProM

Fig. 16. Translating the original process description into a Petri net

are the exact differences?", "What are the most frequently followed paths per process?", and "What does the model describing the current situation look like?". While the first three question could be answered using the conformance checking techniques presented in this paper, the last question was addressed using genetic process mining algorithms (see [15] for further details with respect to this case study).

As a first step, domain experts (i.e., employees of the town hall) helped us to understand the semantics of their initial model, so that it could be translated into a Petri net model, which can be analyzed by the Conformance Checker in the ProM framework (see Figure 16(b)). Note that the crossed-out tasks were not considered because they are executed by external third parties (and not by the personnel from the municipality). Then, an extract of the corresponding log data was exported from the town hall's database and converted[9] into the MXML format, which can be interpreted by the ProM framework. Finally, all fragmentary process instances were removed from the log. This means that only those cases were considered which actually executed both the start activity and one of the possible end activities of the process.

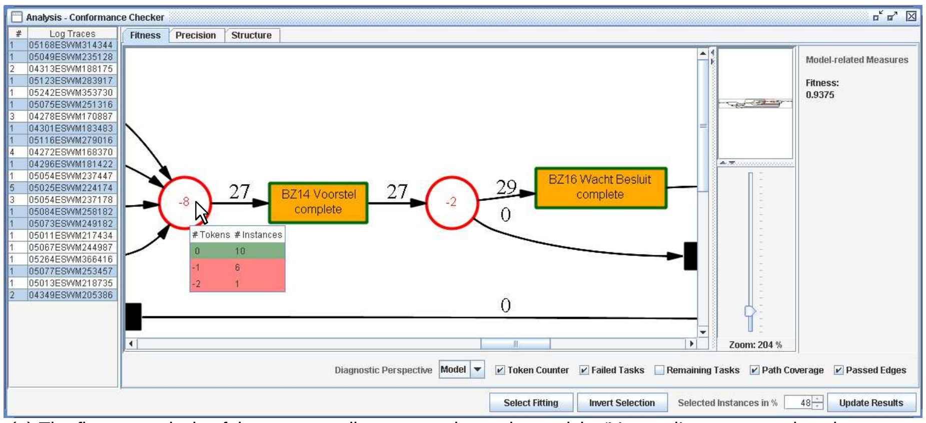
[9] For this, we built a custom plug-in for the ProM*import* framework [21], which converts logs from a wide variety of systems to the XML format used by ProM. It can be downloaded from *www.processmining.org*.

The Petri net models and the cleaned event logs could then be analyzed by the Conformance Checker. Although the municipality uses a workflow system and, therefore, in principle all cases should comply with the prescriptive models, only for one of the four processes all cases were indeed 100% compliant with the original, deployed model (this was one of the three complaint handling processes and the analysis included 358 cases). For example, for the building permit handling procedure only 80% of the 407 cases were fully compliant. The main reason for these deviations was a temporary misconfiguration of the system. When the process was initially configured, a synchronization task could already be executed as soon as 3 out of the 4 incoming parallel branches were ready, which then also happened for some cases. The system administrator inspected the cases that had this problem and confirmed that they happened before this configuration error was corrected. For the complaint handling process depicted in Figure 16 only 51% of the 35 cases were fully compliant with the deployed model. The conformance analysis results for this process are now described in more detail.

Figure 17(a) shows the Fitness result screen of the Conformance Checker. As discussed before, during the replay of the log in the model there may be tokens missing and remaining. Hovering over a problematic place or transition provides more detailed information, e.g., about the number of instances leaving or lacking a certain amount of tokens at that place. Further visualization options indicate the number of times each edge has been passed during log replay, and mark those transitions that have been fired at least once (path coverage). This way, one can directly see how often certain paths in the model were actually used. Note that pressing the *Select Fitting* button automatically selects all traces of the log that are 100% compliant with the given model. This functionality is important for practical use as it enables the separation of fitting and non-fitting process instances. This is particularly useful for large log files. Every subset of the event log can then be further investigated ("Do the non-compliant cases usually take a certain path?" etc.), and may be exported, e.g., to carry out an in-depth performance analysis.

In Figure 17(a) the fitness analysis results of the non-compliant cases of the complaint handling process (the relevant part of the process model is indicated by rectangle A in Figure 16(a) and (b)) are depicted, which shows that activity "Voorstel"[10] was not ready to be executed once (as one token was missing) for six cases, and for one case even twice (as two tokens were missing). Figure 17(b) depicts a screenshot of the log view, which reveals that activity "Voorstel" was sometimes indeed executed two and even three times, which is not allowed according to the original model. So, we were curious to find out how these deviations were possible, and the people responsible for these processes in the municipality explained that the discrepancies resulted from an explicit change of the case by the system administrator. So for example, users had to re-edit an already completed case, or needed to jump to other tasks in the process (and

[10] Note that an understanding of the process is not needed. Therefore, we did not translate the Dutch task names.



#	Log Traces
1	05168ESWM314344
1	05049ESWM235128
2	04313ESWM188175
1	05123ESWM283917
1	05242ESWM353730
1	05075ESWM251316
3	04278ESWM170887
1	04301ESWM183483
1	05116ESWM279016
4	04272ESWM168370
1	04296ESWM181422
1	05054ESWM237447
5	05025ESWM224174
3	05054ESWM237178
1	05084ESWM258182
1	05073ESWM249182
1	05011ESWM217434
1	05067ESWM244987
1	05264ESWM366416
1	05077ESWM253457
1	05013ESWM218735
2	04349ESWM205386

# Tokens	# Instances
0	10
-1	6
-2	1

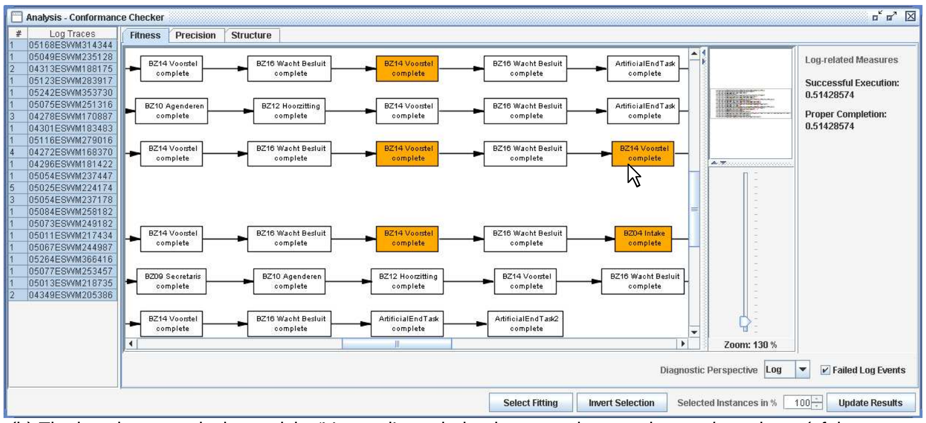
Model-related Measures

Fitness:
0.9375

Diagnostic Perspective: Model
☑ Token Counter ☑ Failed Tasks ☐ Remaining Tasks ☑ Path Coverage ☑ Passed Edges

Selected Instances in %: 48

(a) The fitness analysis of the non-compliant cases shows that activity 'Voorstel' was not ready to be executed *once* for six, and even *twice* for one of the cases


Analysis - Conformance Checker
Fitness
Precision
Structure
BZ14 Voorstel complete
BZ16 Wacht Besluit complete
BZ14 Voorstel complete
BZ16 Wacht Besluit complete
ArtificialEndTask complete
BZ10 Agenderen complete
BZ12 Hoorzitting complete
BZ14 Voorstel complete
BZ16 Wacht Besluit complete
ArtificialEndTask complete
BZ14 Voorstel complete
BZ16 Wacht Besluit complete
BZ14 Voorstel complete
BZ16 Wacht Besluit complete
BZ14 Voorstel complete
BZ14 Voorstel complete
BZ16 Wacht Besluit complete
BZ14 Voorstel complete
BZ16 Wacht Besluit complete
BZ04 Intake complete
BZ09 Secretaris complete
BZ10 Agenderen complete
BZ12 Hoorzitting complete
BZ14 Voorstel complete
BZ16 Wacht Besluit complete
BZ14 Voorstel complete
BZ16 Wacht Besluit complete
ArtificialEndTask complete
ArtificialEndTask2 complete
Zoom: 130 %
Select Fitting
Invert Selection
Update Results

#	Log Traces
1	05168ESWM314344
1	05049ESWM235128
2	04313ESWM188175
1	05123ESWM283917
1	05242ESWM353730
1	05075ESWM251316
3	04278ESWM170887
1	04301ESWM183483
1	05116ESWM279016
4	04272ESWM168370
1	04296ESWM181422
1	05054ESWM237447
5	05025ESWM224174
3	05054ESWM237178
1	05084ESWM258182
1	05073ESWM249182
1	05011ESWM217434
1	05067ESWM244987
1	05264ESWM366416
1	05077ESWM253457
1	05013ESWM218735
2	04349ESWM205386

Log-related Measures

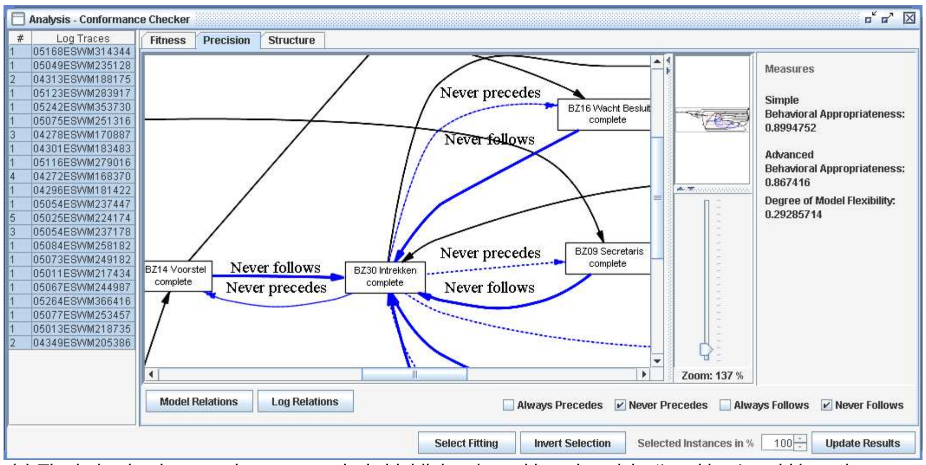
Successful Execution:
0.51428574

Proper Completion:
0.51428574

Diagnostic Perspective: Log
☑ Failed Log Events

Selected Instances in %: 100

(b) The log view reveals that activity 'Voorstel' was indeed executed *two* and even *three* times (cf. log trace indicated by mouse pointer) by some of the cases, which is not allowed according to the original model



#	Log Traces
1	05168ESWM314344
1	05049ESWM235128
2	04313ESWM188175
1	05123ESWM283917
1	05242ESWM353730
1	05075ESWM251316
3	04278ESWM170887
1	04301ESWM183483
1	05116ESWM279016
4	04272ESWM168370
1	04296ESWM181422
1	05054ESWM237447
5	05025ESWM224174
3	05054ESWM237178
1	05084ESWM258182
1	05073ESWM249182
1	05011ESWM217434
1	05067ESWM244987
1	05264ESWM366416
1	05077ESWM253457
1	05013ESWM218735
2	04349ESWM205386

Measures

Simple Behavioral Appropriateness:
0.8994752

Advanced Behavioral Appropriateness:
0.867416

Degree of Model Flexibility:
0.29285714

☐ Always Precedes ☑ Never Precedes ☐ Always Follows ☑ Never Follows

Selected Instances in %: 100

(c) The behavioral appropriateness analysis highlights that, although activity 'Intrekken' could have been executed in many states of the process, this was not used for the recorded cases

Fig. 17. Screenshots while analyzing the complaint handling process

the deployed model did not allow for this). This is a very interesting result as it shows that people may need to deviate from prescribed procedures even if they have a workflow system guiding that process in a non-flexible way. Then, the changes are realized in an ad-hoc way, such as through a system administrator who has the right to work "behind the back" of the system.

Figure 17(c) shows a screenshot of the behavioral appropriateness analysis of the same process. As explained earlier, the visualization indicates where the log becomes more specific than the model (i.e., activities *always* or *never* followed or preceded each other while according to the model this is not required). In Figure 17(c), the activity in the center of the diagnostic visualization is activity "Intrekken" (indicated by rectangle B in Figure 16(a) and (b)), which is connected to almost every other activity in the process as it relates to some *cancel* activity (i.e., when the complaint handling is not further pursued). Analyzing the behavior in the log, it became clear that complaints were only cancelled in a very early stage of the process, so that after most of the other activities activity "Intrekken" never happened (although this would be possible with respect to the model). This is not necessarily a problem but provides insight into the way the process is actually executed.

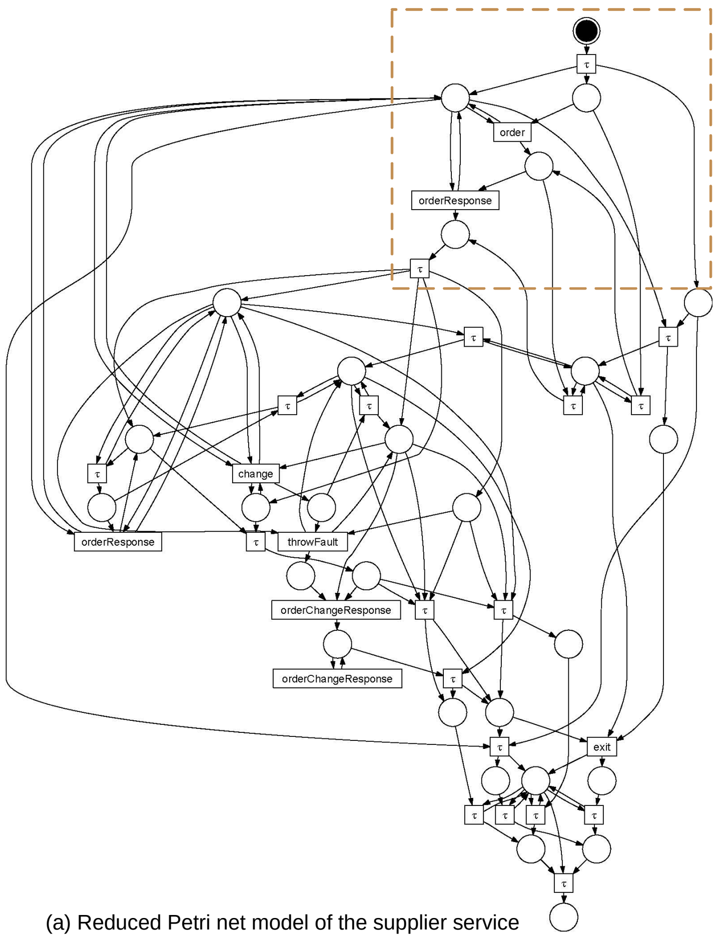
These examples demonstrate that using the presented techniques it is possible to discover conformance problems in real-life scenarios.

8.2 Conformance Checking of Service Behavior

In [4] we investigated conformance in the context of service-oriented systems, which are composed of services that are typically (a) independently developed and operated, and (b) interact with one another exclusively through message exchanges. To coordinate the communication between different services there are process descriptions that specify how these services should interact. Since partners will typically not expose the internal structure and state of their services, the question of conformance arises: "Do all parties involved operate as described?". The expected behavior may deviate as soon as, e.g., a service receives a reply of the wrong type, messages are received in the wrong order, etc. Using abstract BPEL as a choreography language, and observing the exchanged SOAP messages we demonstrated that it is possible to tackle this problem using the conformance checking techniques presented in this paper.

The example of a simple supplier service was used in the following way. First, the supplier service was specified as an abstract BPEL process, which is a non-executable process specification describing the business protocol as seen from one of the partners involved. Second, we automatically created a Petri net description of the intended choreography, using the translation described in [28] and implemented in the tool BPEL2PNML[11]. Finally, the resulting net was reduced using a tool called WofBPEL, which yielded a process model that

[11] Both documentation and software can be downloaded from the BABEL project pages *http://www.bpm.fit.qut.edu.au/projects/babel/tools/*.



(a) Reduced Petri net model of the supplier service

	Scenario	Fitness	Log trace
desirable behavior	1	1.0	(order, orderResponse)
	2	1.0	(order, orderResponse, orderResponse, orderResponse)
	3	1.0	(order, orderResponse, change, orderChangeResponse)
	4	1.0	(order, orderResponse, orderResponse, change, orderChangeResponse)
	5	1.0	(order, orderResponse, change, orderChangeResponse, orderChangeResponse)
undesirable behavior	6	0.625	(order)
	7	0.749	(order, orderResponse, change)
	8	0.905	(orderResponse)
	9	1.0	(order, orderResponse, change, orderResponse, orderChangeResponse)
	10	0.759	(order, change, orderChangeResponse)
	11	0.0	(change)
	12	0.914	(order, orderResponse, change, orderChangeResponse, change)
	13	0.971	(order, orderResponse, change, change, orderChangeResponse)

(b) Desirable and undesirable scenarios for the supplier service execution

Fig. 18. Supplier service example

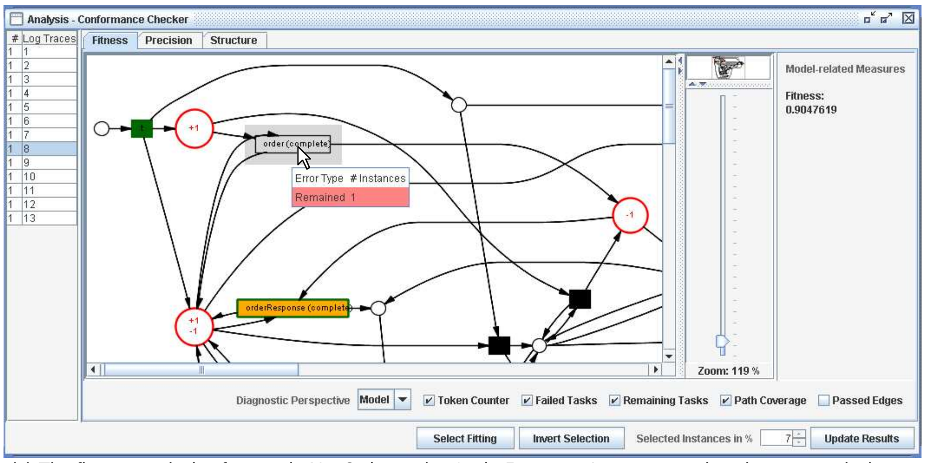
could be analyzed by the Conformance Checker in the ProM framework (see Figure 18(a)).

In [4] we showed that it is possible to monitor and correlate messages, and to group them into log traces where each trace reflects one concrete service execution. By implementing the example process in Oracle BPEL we could obtain logs of SOAP messages for both directions (both to and from the supplier service). Note that we do not make assumptions with respect to the implementation of the process logic in the service, instead of executable BPEL any other language could have been used.

Having demonstrated that it is feasible to obtain such event log from real service executions we then used the presented conformance checking techniques to validate the supplier service specification for a number of interaction scenarios. Figure 18(b) shows five execution sequences which should be valid for the supplier service (Scenarios 1 – 5) and eight which should not (Scenarios 6 – 9 correspond to possible violations by the supplier service and 10 – 13 contain violations by the client or environment of the service). Importing the reduced Petri net model generated from the abstract BPEL process, the Conformance Checker can replay the given scenarios in this model, and the fitness measurement indicates whether a scenario corresponds to a possible execution sequence for that process.

Consider for example Figure 19(a), in which the Conformance Checker shows the dotted part of the model in Figure 18(b) after the replay of Scenario 8. In this situation a single *orderResponse* was sent without having received any previous *order*, which is not allowed. Following the control flow of the model it can be observed that the *order* transition is supposed to fire first to produce a token in the enlarged place on the right, which can be consumed by the *orderResponse* transition afterwards. However, since the log replay is carried out from a log-based perspective the missing tokens (indicated by a − sign) are created artificially and the task belonging to the observed message in the model (i.e., the *orderResponse* transition) is executed immediately. The fact that it had been forced to do so is recorded and the task is marked as having failed successful execution (i.e., it was not enabled). Furthermore, there are tokens remaining in the enlarged places in the upper and the lower left corner (indicated by a + sign), which leads to the *order* transition remaining enabled after replay has finished. Remaining tasks are visualized with the help of a shaded rectangle in the background and they point to situations where a task was expected to be executed but did not happen.

Now reconsider Figure 18(b), where the Fitness column indicates for each scenario whether it corresponds to a valid execution sequence for our supplier service (i.e., fitness = 1.0) or not (i.e., fitness < 1.0). As it shows 100 % fitness for Scenario 1 – 5 the abstract BPEL process has been proven to be a valid specification with respect to the “well-behaving” conversation scenarios we thought of. However, it also allows for an execution sequence that we classified as undesirable behavior, namely Scenario 9: Although another *orderResponse* is sent after a *change* request has been received already (and thus only *orderChangeRe-*



Error Type # Instances
Remained 1

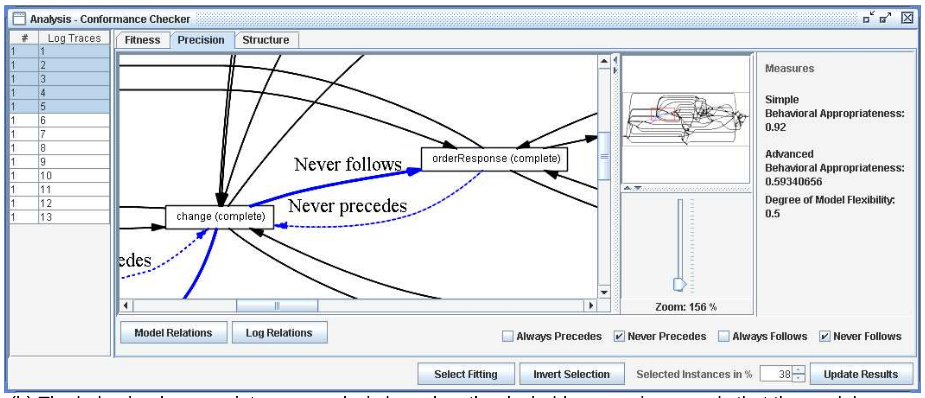
Model-related Measures

Fitness:
0.9047619

Diagnostic Perspective: Model
[✓] Token Counter [✓] Failed Tasks [✓] Remaining Tasks [✓] Path Coverage [ ] Passed Edges

Selected Instances in %: 7

(a) The fitness analysis of scenario No. 8 shows that 'orderResponse' was not ready to be executed when it occurred (tokens were missing), and that 'order' was expected to occur but did not happen (tokens were remaining)



Measures

Simple
Behavioral Appropriateness:
0.92

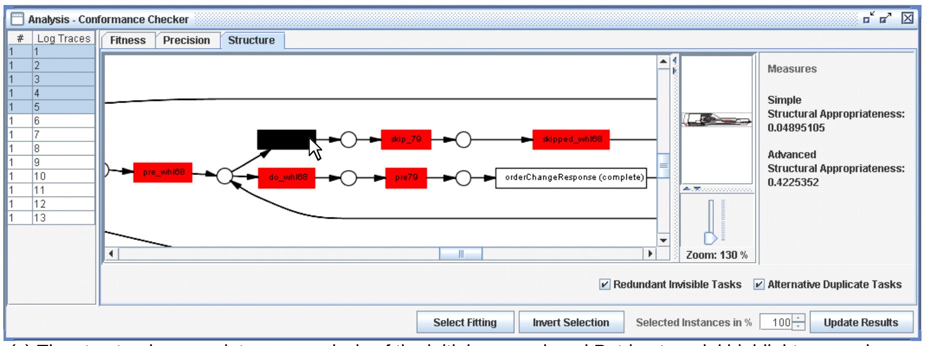
Advanced
Behavioral Appropriateness:
0.59340656

Degree of Model Flexibility:
0.5

[ ] Always Precedes [✓] Never Precedes [ ] Always Follows [✓] Never Follows

Selected Instances in %: 38

(b) The behavioral appropriateness analysis based on the desirable scenarios reveals that the model allows for more behavior than expected. Due to intermediate states it is possible to send an 'orderResponse' after a 'change' request has been received



Measures

Simple
Structural Appropriateness:
0.04895105

Advanced
Structural Appropriateness:
0.4225352

[✓] Redundant Invisible Tasks [✓] Alternative Duplicate Tasks

Selected Instances in %: 100

(c) The structural appropriateness analysis of the initial, non-reduced Petri net model highlights a number of redundant invisible tasks (in the visualized fragment all invisible tasks except the one indicated by the mouse pointer are redundant)

Fig. 19. Screenshot while analyzing the service execution scenarios

sponses should be sent) the scenario proved to comply with the given abstract BPEL process specification. This was an interesting result as it made us aware of the fact that—due to a number of intermediate states—the chosen fault/event handler construct did not completely capture the intended constraint. The same conclusion can be drawn from the behavioral appropriateness analysis based on the five desirable scenarios, where the result is depicted in Figure 19(b).

Finally, Figure 19(c) depicts a screenshot from the structural appropriateness analysis of the non-reduced Petri net model. Recall that in the process of generating a Petri net from the BPEL model, the tool WofBPEL was used to reduce the initially created Petri net. In fact, a part of this reduction is the removal of redundant invisible tasks as described in this paper. The reduced model (i.e., after applying WofBPEL) contains 27 transitions, 28 places, and 121 arcs. The non-reduced model (which is partly shown in Figure 19(c)) contains 71 transitions, 72 places, and 209 arcs, i.e., 41 invisible transitions are redundant and can be removed. These redundant invisible tasks are also detected and highlighted by the Conformance Checker. Note that, besides the export of all the diagnostic visualizations, the Conformance Checker also offers the export of the reduced Petri net model (i.e., an equivalent model without the redundant invisible tasks that were detected).

This application demonstrates that behavioral appropriateness analysis can help to detect undesirable behavior, and that in the presence of negative examples (i.e., "forbidden" scenarios) the fitness metric can also be used to analyze the behavioral appropriateness of a given process model.

9 Related Work

Conformance checking as presented in this paper is closely related to the work of Cook et al. [14, 13] who have introduced the concept of process validation. They propose a technique comparing the event stream coming from the process model with the event stream from the execution log based on two different string distance metrics. To address the problem of time-complexity while exploring the state space of the model they investigate (and reject) several techniques from domains like compiler research and regular-expression matching. In the end an incremental, data-driven state-space search is suggested, using heuristics to reduce the cost. An interesting point is that they include the possibility to assign weights to differentiate the relative importance of specific types of events. In [13] the results are extended to include time aspects. The notion of conformance has also been discussed in the context of security [6], business alignment [1], and genetic mining [7] (all proposing some kind of replay). However, in each of the papers mentioned only fitness is considered and appropriateness is mostly ignored. (Note that more recent work on genetic mining also includes "penalties" for "too much behavior" [25, 15].) In [36] case-based reasoning is applied to explicitly record information about non-compliant cases, which can be re-used for potential adaptations of the business process model. Finally, this paper builds on preliminary work reported in [32, 31].

Conformance checking assumes the presence of a given descriptive or prescriptive process model, and therefore has a different starting point, but nevertheless it is closely related to typical process mining techniques [10, 9], which aim at the discovery of a process model based on some event log. In the desire to derive a "good" model for the behavior observed in the execution log, shared notions of fitness, behavioral appropriateness and structural appropriateness can be identified. In [18] the process mining problem is faced with the aim of deriving a model which is as compliant as possible with the log data, accounting for fitness (called completeness) and also behavioral appropriateness (called soundness). Starting with a disjunctive workflow schema containing all the traces from the log (cf. Figure 4(b)) they try to incrementally cluster these traces until a given lower bound for the number of schemata contained is reached, which, in fact, corresponds to some notion of structural appropriateness as well. Another example is the process mining approach presented in [37], which aims at the discovery of a WF-net that (i) potentially generates all event sequences appearing in the execution log (i.e., fitness), (ii) generates as few event sequences not contained in the execution log as possible (i.e., behavioral appropriateness), and (iii) captures concurrent behavior and (iv) is as simple and compact as possible (i.e., structural appropriateness). Moreover, techniques such as considering some form of causal relation can be borrowed from the process mining research, just as insights gained into concepts like correctness, completeness, and noise are also relevant in the context of conformance checking.

Related to conformance checking as presented in this paper is the checking of a temporal formula with respect to a log [2]. In [2] we show that the LTL Checker in ProM can check which cases satisfy a given property. Furthermore, in [22] the authors present a polynomial algorithm to decide whether a scenario (given as a labelled partial order) is executable in a given Petri net ([12] presents the VipTool, which implements the approach). Both approaches are complementary to the approach presented here.

Process mining and conformance checking can be seen in the broader context of Business (Process) Intelligence (BPI) and Business Activity Monitoring (BAM). In [19, 20] a BPI tool set on top of HP's Process Manager is described. The BPI tool suite includes a so-called "BPI Process Mining Engine". In [26] Zur Muehlen describes the PISA tool which can be used to extract performance metrics from workflow logs. Similar diagnostics are provided by the ARIS Process Performance Manager (PPM). The latter tool is commercially available and a customized version of PPM is the Staffware Process Monitor (SPM), which is tailored towards mining Staffware logs. Note that none of the latter tools is supporting conformance checking. The focus of these tools is often on performance measurements rather than monitoring (un)desirable behavior.

10 Conclusion

From the coexistence of explicit process models and event logs originates the interesting question "Do the model and the log *conform* to each other?". This question is highly relevant for all kinds of situations where there is a notion of a process model but people can deviate. This paper proposes an incremental approach to check the conformance of a process model and an event log. At first, the *fitness* between the log and the model needs to be ensured (i.e., "Does the observed process comply with the control flow specified by the process model?"). At second, the *appropriateness* of the model can be analyzed with respect to the log (i.e., "Does the model describe the observed process in a suitable way?"). During this second phase two aspects of appropriateness are considered, evaluating *structural* properties of the process model on the one hand ("Is the behavior specified by the model represented in a structurally suitable way?") and *behavioral* properties on the other ("Does the model specify the behavior of the observed process in a sufficiently specific way?").

One metric (f) has been defined to address fitness. Moreover, two metrics for structural appropriateness (a_S and a'_S) and two metrics for behavioral appropriateness (a_B and a'_B) have been defined. Together they allow for the quantification of conformance. An evaluation of properties of previous metrics has led to the development of the two new appropriateness metrics a'_S and a'_B, which—in contrast to their counterparts a_S and a_B—are able to measure only one dimension of appropriateness, independently of the other (i.e., they are *stable* and orthogonal). Furthermore, they indicate when an "optimal" solution has been reached (i.e., they are *analyzable*). Besides the quantification of fitness and appropriateness, it is crucial to assist the analyst in finding the location of a conformance problem. It has been shown that we are also able to locate the respective problem areas in the model or the log. Two different applications (Town Hall in the Netherlands and the Oracle BPEL application) of the implemented Conformance Checker demonstrated that the presented techniques constitute a powerful means to indicate a conformance problem and to estimate its dimension, while providing the user with some visual feedback pinpointing those parts that should be revised.

Future work will aim at the development of new techniques for both measuring and visualizing non-conformance, and at the support of further modeling languages. Note that other process modeling languages may include special constructs, which can be exploited by dedicated conformance checking techniques. An example is the OR-split/join in EPCs, where an arbitrary number of branches can be activated/joined during log replay. Furthermore, the structural appropriateness methods are naturally biased by the used modeling notation. Also note that the availability of suitable log data will be crucial for any future system that is used to support business processes because this enables a systematic analysis of how these processes are actually carried out. If additional information is present in the log and the specification, even further perspectives on the business process (such as the data, performance, and organizational perspective) could be exploited to verify data dependency constraints, whether deadlines were met, or whether the resource specifications were respected.

Acknowledgements

This research is supported by the Technology Foundation STW, EIT, the EU project SUPER, and the IOP program of the Dutch Ministry of Economic Affairs. Furthermore, the authors would like to thank all ProM developers for their on-going work on process mining techniques.

References

1. W.M.P. van der Aalst. Business Alignment: Using Process Mining as a Tool for Delta Analysis. In J. Grundspenkis and M. Kirikova, editors, *Proceedings of the 5th Workshop on Business Process Modeling, Development and Support (BPMDS'04)*, volume 2 of *Caise'04 Workshops*, pages 138–145. Riga Technical University, 2004.
2. W.M.P. van der Aalst, H.T. de Beer, and B.F. van Dongen. Process Mining and Verification of Properties: An Approach based on Temporal Logic. In R. Meersman and Z. Tari et al., editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005*, volume 3760 of *Lecture Notes in Computer Science*, pages 130–147. Springer-Verlag, Berlin, 2005.
3. W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. ProM 4.0: Comprehensive Support for Real Process Analysis. In J. Kleijn and A. Yakovlev, editors, *Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007)*, volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer-Verlag, Berlin, 2007.
4. W.M.P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and H.M.W. Verbeek. Choreography Conformance Checking: An Approach based on BPEL and Petri Nets (extended version). BPM Center Report BPM-05-25, BPMcenter.org, 2005 (To appear in ACM Transactions on Internet Technology, special issue on Middleware for Service-oriented Computing).
5. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems.* MIT press, Cambridge, MA, 2002.
6. W.M.P. van der Aalst and A.K.A. de Medeiros. Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance. In N. Busi, R. Gorrieri, and F. Martinelli, editors, *Second International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004)*, pages 69–84. STAR, Servizio Tipografico Area della Ricerca, CNR Pisa, Italy, 2004.
7. W.M.P. van der Aalst, A.K.A. de Medeiros, and A.J.M.M. Weijters. Genetic Process Mining. In G. Ciardo and P. Darondeau, editors, *26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005)*, volume 3536 of *Lecture Notes in Computer Science*, pages 48–69. Springer-Verlag, Berlin, 2005.
8. W.M.P. van der Aalst and M. Song. Mining Social Networks: Uncovering Interaction Patterns in Business Processes. In J. Desel, B. Pernici, and M. Weske, editors, *International Conference on Business Process Management (BPM 2004)*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer-Verlag, Berlin, 2004.
9. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.

10. W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.
11. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
12. R. Bergenthum, J. Desel, G. Juhás, and R. Lorenz. Can I Execute My Scenario in Your Net? VipTool Tells You! In S. Donatelli and P.S. Thiagarajan, editors, *Petri Nets and Other Models of Concurrency - ICATPN 2006*, volume 4024 of *Lecture Notes in Computer Science*, pages 381–390. Springer-Verlag, Berlin, 2006.
13. J.E. Cook, C. He, and C. Ma. Measuring Behavioral Correspondence to a Timed Concurrent Model. In *Proceedings of the 2001 International Conference on Software Mainenance*, pages 332–341, 2001.
14. J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176, 1999.
15. A.K. Alves de Medeiros. *Genetic Process Mining.* PhD thesis, Department of Technology Management, Technical University Eindhoven, 2006.
16. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, Cambridge, UK, 1995.
17. J. Desel, W. Reisig, and G. Rozenberg, editors. *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2004.
18. G. Greco, A. Guzzo, L. Pontieri, and D. Saccá. Mining Expressive Process Models by Clustering Workflow Traces. In *Proc of Advances in Kowledge Discovery and Data Mining, 8th Pacific-Asia Conference (PAKDD 2004)*, pages 52–62, 2004.
19. D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.C. Shan. Business process intelligence. *Computers in Industry*, 53(3):321–343, 2004.
20. D. Grigori, F. Casati, U. Dayal, and M.C. Shan. Improving Business Process Quality through Exception Understanding, Prediction, and Prevention. In P. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. Snodgrass, editors, *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 159–168. Morgan Kaufmann, 2001.
21. C.W. Günther and W.M.P. van der Aalst. A Generic Import Framework for Process Event Logs. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops, Workshop on Business Process Intelligence (BPI 2006)*, volume 4103 of *Lecture Notes in Computer Science*, pages 81–92. Springer-Verlag, Berlin, 2006.
22. G. Juhás, R. Lorenz, and J. Desel. Can I Execute My Scenario in Your Net? In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 289–308. Springer-Verlag, Berlin, 2005.
23. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation.* Addison-Wesley, Reading MA, 1998.
24. P. Liggesmeyer. *Software-Qualität – Testen, Analysieren und Verifizieren von Software.* Spektrum Akademischer Verlag, Heidelberg, Berlin, 2002.
25. A.K.A. de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic Process Mining: A Basic Approach and its Challenges. In M. Castellanos and T. Weijters, editors, *First International Workshop on Business Process Intelligence (BPI'05)*, pages 46–57, Nancy, France, September 2005.

26. M. zur Mühlen and M. Rosemann. Workflow-based Process Monitoring and Controlling - Technical and Organizational Issues. In R. Sprague, editor, *Proceedings of the 33rd Hawaii International Conference on System Science (HICSS-33)*, pages 1–10. IEEE Computer Society Press, Los Alamitos, California, 2000.
27. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
28. C. Ouyang, W.M.P. van der Aalst, S. Breutel, M. Dumas, A.H.M. ter Hofstede, and H.M.W. Verbeek. Formal Semantics and Analysis of Control Flow in WS-BPEL. BPM Center Report BPM-05-13, BPMcenter.org, 2005.
29. J.L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, Englewood Cliffs, 1981.
30. M. Reichert and P. Dadam. ADEPTflex - Supporting Dynamic Changes of Workflows Without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
31. A. Rozinat and W.M.P. van der Aalst. Conformance Testing: Measuring the Alignment Between Event Logs and Process Models. BETA Working Paper Series, WP 144, Eindhoven University of Technology, Eindhoven, 2005.
32. A. Rozinat and W.M.P. van der Aalst. Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In C. Bussler et al., editor, *Business Process Management 2005 Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 163–176. Springer-Verlag, Berlin, 2006.
33. P. Sarbanes, G. Oxley, and et al. Sarbanes-Oxley Act of 2002, 2002.
34. A.W. Scheer. *ARIS: Business Process Modelling*. Springer-Verlag, Berlin, 2000.
35. H.M.W. Verbeek. *Verification and Enactment of Workflow Management Systems*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2004.
36. B. Weber, M. Reichert, S. Rinderle, and W. Wild. Towards a Framework for the Agile Mining of Business Processes. In C. Bussler et al., editor, *Business Process Management 2005 Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 191–202, Nancy, France, September 2006. Springer-Verlag, Berlin.
37. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data. In *Proceedings of the Third International NAISO Symposium on Engineering of Intelligent Systems (EIS 2002)*, pages 65–65. NAISO Academic Press, Sliedrecht, The Netherlands, 2002.