Conformance Checking of Service Behavior (Revised Version)

W.M.P. van der Aalst[1,2], M. Dumas[2], C. Ouyang[2], A. Rozinat[1], and H.M.W. Verbeek[1]

[1] Department of Information Systems, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{w.m.p.v.d.aalst,a.rozinat,h.m.w.verbeek}@tue.nl
[2] Faculty of Information Technology, Queensland University of Technology,
GPO Box 2434, Brisbane QLD 4001, Australia
{c.ouyang,m.dumas}@qut.edu.au

Abstract A service-oriented system is composed of independent software units, namely services, that interact with one another exclusively through message exchanges. The proper functioning of such system depends on whether or not each individual service behaves as the other services expect it to behave. Since services may be developed and operated independently, it is unrealistic to assume that this is always the case. This paper addresses the problem of checking and quantifying how much the actual behavior of a service, as recorded in message logs, conforms to the expected behavior as specified in some process model. Concretely, we consider the case where the expected behavior is defined using the de-facto industry standard BPEL (i.e., the Business Process Execution Language for Web Services). BPEL process definitions are translated into Petri nets and Petri net-based conformance checking techniques are applied to quantify two complementary indicators of conformance: *fitness* and *appropriateness*. The approach is supported by ProM (a toolset for business process analysis and mining) and has been applied in a setting using multiple Oracle BPEL servers.

1 Introduction

A service-oriented system is composed of services that interact with one another for a given purpose. To ensure that this purpose is attained, designers specify these interactions and their dependencies in some form. In principle, the participating services are implemented, adapted, or configured to comply with this specification. However, services may be developed, operated, and evolved by independent teams or organizations. Thus, there is no guarantee that once a system is under operation, some services will not deviate from the specification. For example, after sending a request, a service may receive a reply of the wrong type, a service may reject a message sent by another service, messages may be received in the wrong order, etc. Furthermore, each of these unexpected behaviors may cascade into other deviations. In general terms, service independence raises the question of *conformance*: "Do all services in a service-oriented system operate as expected?".

This paper addresses the following question: Given an expected service behavior captured as one or several *process models*, and an observed behavior as registered in a *message log*, does the observed behavior conform to the expected behavior?

We use the term *service choreography* to refer to a specification of the expected behavior of an individual service or collection of services. Choreographies may be captured in a number of languages. In this paper, we consider a standard for service behavior specification, namely the Business Process Execution Language for Web Services (BPEL) [14],[3] but the results can well be applied to other languages. Also, the paper assumes that messages are represented according to the XML and SOAP standards [16], but the proposed techniques could be applied to other message formats. Finally, the paper focuses on checking the fulfillment of control flow dependencies captured in the choreography. We do not consider the issue of checking whether or not each individual message conforms to its expected message type as this is a well-understood problem.

When a choreography and a message log do not conform, two scenarios are possible. First of all, the model may be assumed to be "correct" because it represents the way partners should work, and the question is whether the events in the log are consistent with the process model. For example, the log may contain event sequences that are not possible according to the model. This may indicate violations of the choreography. Second, the event log may be assumed to be "correct" because it is what really happened. In the latter case the question is whether the choreography that has been agreed upon is no longer valid and should be modified. In this paper, we provide techniques for addressing both of the above scenarios.

To illustrate the need for conformance checking in the context of services we briefly discuss some examples:

- Consider a travel agent offering complete vacation packages to its customers. To offer this service it uses a wide variety of service providers (airlines, car rental companies, bus companies, hotel chains, etc) to "assemble" interesting vacation packages. More-

[3] Specifically, choreographies can be captured using one or several *BPEL abstract processes*, that is, BPEL processes that are not necessarily executable.

over, the travel agent is also using services of insurance companies and credit card companies. This is only possible if all services behave as expected. Since each of the parties involved is autonomous, there is no way to enforce conformance. However, by monitoring all interactions one can measure conformance and locate violations.

- Consider an electronic bookstore where customers can order books, music, and movies. Note that one customer can order multiple items. For each of the items the bookstore accesses a service of some content provider (e.g., a publisher, music company, or movie company). Moreover, since the customer can select different payment methods and different ways of shipping the items, the electronic bookstore needs to use different payment and shipment services. Note that not all items may become available at the same time, e.g., some items can be shipped in one or two days while others may take weeks. Hence, one customer order may result in different shipments. Again the parties involved are autonomous and may deviate from some agreed-upon service specification.
- Later in this paper, we will consider the interactions between a supplier and one of its customers. The customer can place an order and several messages are exchanged to process the order. Moreover, the customer can change the order under some circumstances. Although only two services are involved many things can go wrong, e.g., a customer may try to change a non-confirmed order or change an already completed order.

The above examples illustrate that there is a link between Service Level Agreements (SLAs) and Quality of Service (QoS) and conformance checking. However, SLAs and QoS metrics typically focus on simple performance indicators such as the time, failure, or costs of activities and processes [17]. Such indicators assume that the process conforms to some predefined model, i.e., the observed behavior is not compared with respect to some specification and the control-flow is not taken into account because things are considered at an aggregated level.

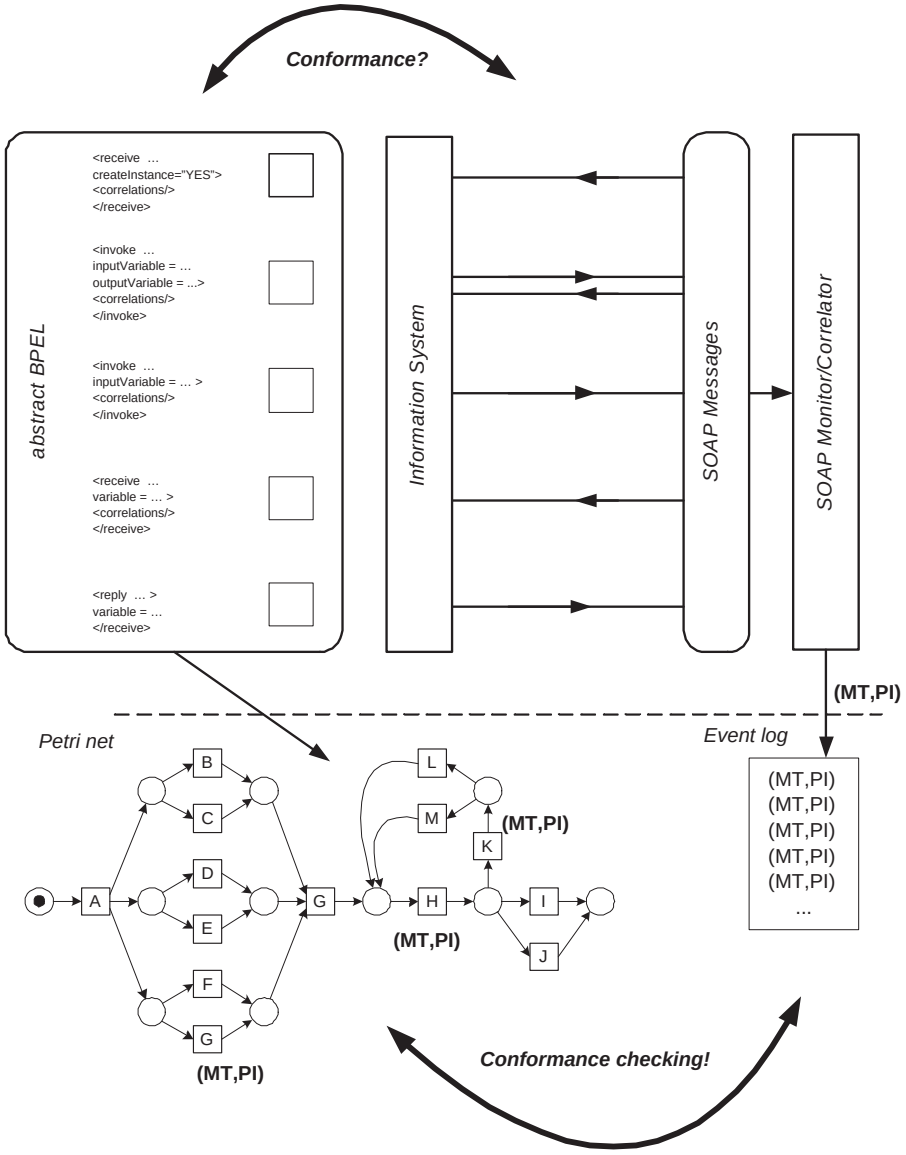


Figure 1. Overview of the approach. The top level shows the process model in BPEL and the recorded behavior in the form of SOAP messages. The BPEL specification is mapped onto Petri nets and the SOAP messages are put in an event log. Finally, both are compared using conformance checking.

Figure 1 describes the approach proposed in this paper. Based on a process model described as an abstract BPEL process, we generate a Petri net [22]. We use a translation described in [44] and implemented in a tool called BPEL2PNML[4]. We also propose an approach to monitor and to correlate SOAP messages in order to construct events logs.

[4] Documentation and software available from `http://www.bpm.fit.qut.edu.au/projects/babel/tools/`.

Conformance checking is performed by comparing the obtained event logs with the Petri net.

The paper considers two notions of conformance: *fitness* and *appropriateness*. An event log and Petri net "fit" if the Petri net can generate each trace in the log. In other words: the Petri net describing the choreography should be able to "parse" every event sequence observed by monitoring e.g. SOAP messages. In [46] it is shown that it is possible to quantify fitness, e.g., an event log and Petri net may have a fitness of 0.66 indicating that 66 percent of the events in the log are possible according to the model. Unfortunately, a good fitness does not imply conformance, e.g., 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. This is why we consider a second dimension, namely appropriateness. Appropriateness captures the idea of *Occam's razor*, i.e., "one should not increase, beyond what is necessary, the number of entities required to explain anything". A model is appropriate if it is the "simplest" one, both structurally and behaviorally, explaining the observed behavior. Thus, overfitting and underfitting models are avoided.

The proposed techniques have been implemented in a tool called *Conformance Checker*. This tool has been integrated into the *ProM framework*[5]. Although ProM offers a wide range of tools related to process mining [9] (e.g., LTL checking, process discovery, verification, etc.), in this paper we focus on ProM's Conformance Checker and its application to monitoring services.

The rest of the paper is organized as follows. Section 2 describes the different settings where conformance checking can be used in a meaningful way. Section 3 elaborates on the notion of conformance and introduces the ProM Conformance Checker. Then we discuss the mapping of BPEL onto WF-nets [44], a subclass of Petri nets [1]. Section 5 discusses ways of extracting high-level event logs from SOAP message logs. Section 6 describes a case study demonstrating the feasibility of our approach and the tools we

[5] Documentation and software available from `http://www.processmining.org`.

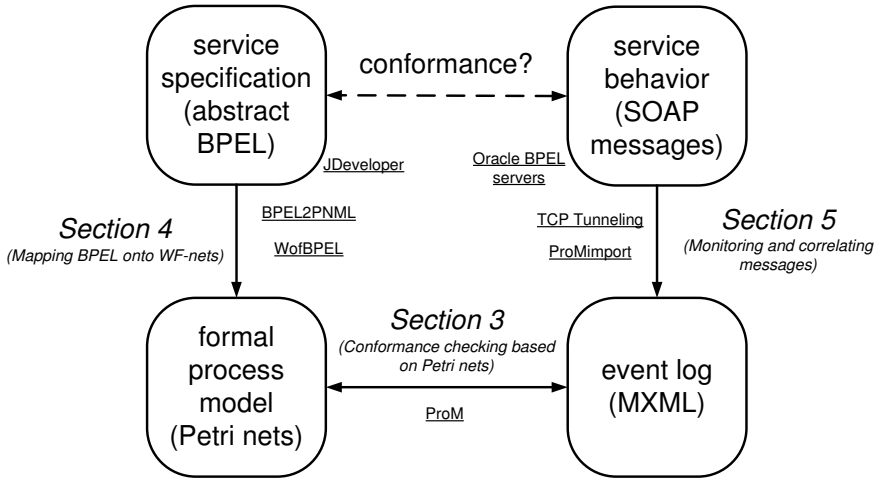


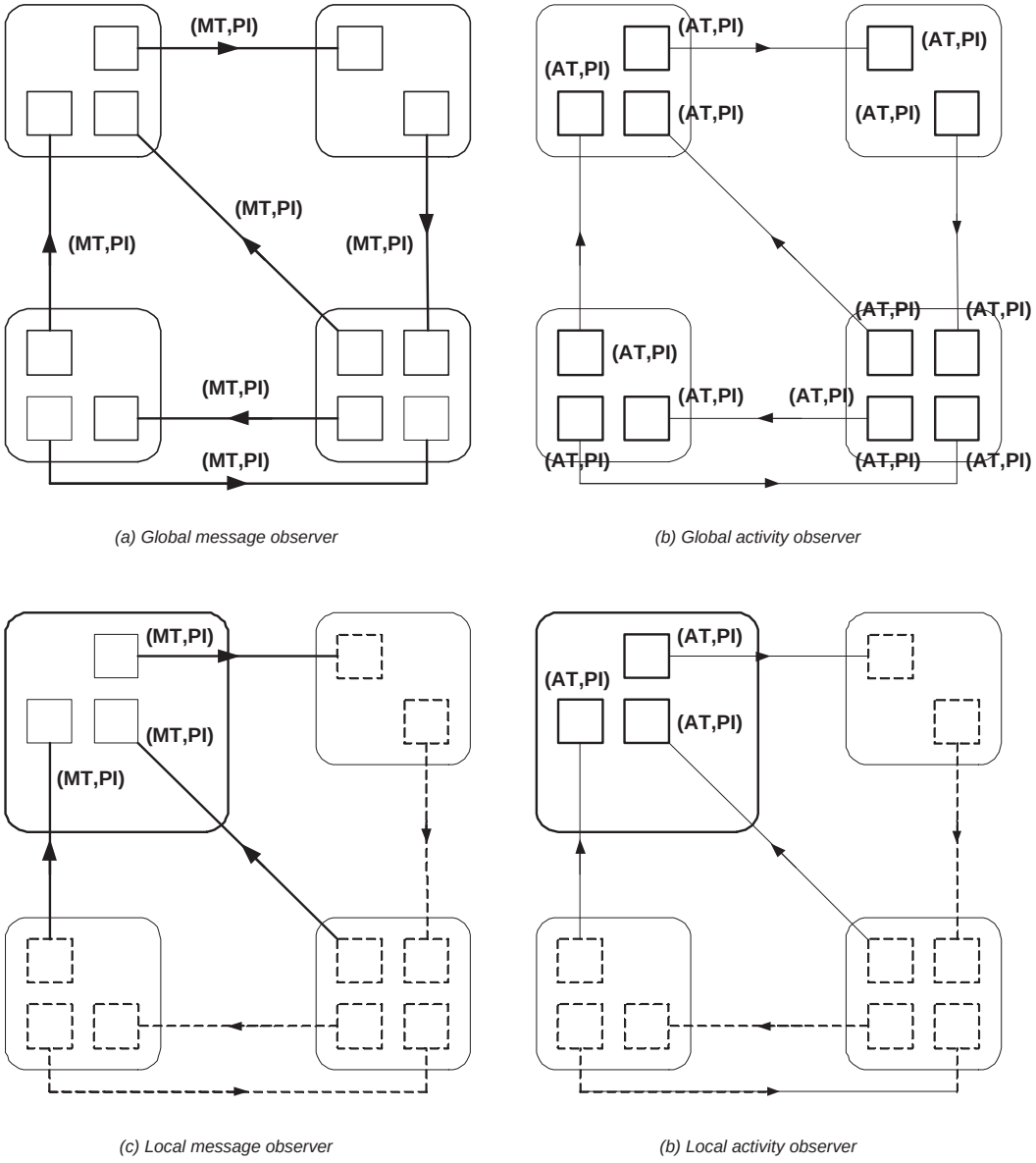
Figure 2. Outline of the paper showing the role of the core sections of the paper and the tools being used.

have developed. Related work is discussed in Section 7 and Section 8 concludes the paper. Figure 2 shows the relationships between the main sections in this paper.

2 Background

We consider four possible settings for choreography conformance checking as illustrated in Figure 3. To illustrate each setting, we consider a Service-Oriented Architecture (SOA) where four services interact by exchanging messages. Each service performs multiple activities and each activity results in certain message exchanges. Depending on the setting, we assume only some activities and messages to be visible (i.e. recorded in logs available for analysis). Visible activities or messages are indicated in bold in the figure.

Figure 3(a) shows the setting where there is a global observer that can monitor all messages. This could be realized by implementing a message broker that connects all services. Figure 3(b) considers another scenario where an ideal observer can monitor any activity performed by each of the involved services. In this context it should be noted that middleware products such as IBM's Websphere [33], Oracle BPEL [43], and Colombo [21], maintain detailed logs of activities. If all the services to be monitored share these logs, it is possible to re-construct a global log of all activities. Figure 3(c) describes the setting where all messages exchanged by a specific service are recorded.



(a) Global message observer *(b) Global activity observer*

(c) Local message observer *(b) Local activity observer*

Figure 3. Four possible settings for choreography conformance checking: (a) relevant messages exchanged between all services involved in a choreography are visible, (b) relevant activities executed inside all services involved in a choreography are visible, (c) relevant messages exchanged with a single service are visible, and (d) relevant activities executed within a single service are visible. (MT = Message Type, AT = Activity Type, and PI = Process Instance).

Here, the observer does not need to view the process globally: a local monitoring facility within one of the services suffices. Finally, Figure 3(d) illustrates the scenario where all the activities performed by a given service are recorded, but no global view of all activities performed by all services is available. Recall that todays middleware products are capable of recording such events.

The four possible settings for choreography conformance checking depicted in Figure 3 put different demands on the functionality present to log events. Clearly, it is easier to only logs event locally, i.e., settings (c) and (d) in Figure 3. Moreover, the monitoring of messages does not impose any constraints on the middleware being used. Therefore, this paper will focus on the setting illustrated by Figure 3(c). However, our approach is more generic and could be easily applied to the other three setting assuming that the required events can be captured.

For conformance checking, it is crucial that each event recorded in the log can be linked to a process instance and a process model element (e.g., an activity in BPEL terms or a transition in Petri-net terms).[6] This is reflected in Figure 3 by *(MT,PI)* and *(AT,PI)*. *PI* refers to a specific process instance, i.e., a unique identifier of the case being processed. Examples of a *PI* are a customer id, customer order reference, a social security number, a patient id, etc. *MT* is the message type that can be linked to some activity and *AT* refers to some activity. Whether an *MT* or *AT* is recorded depends on the setting (cf. Figure 3). Examples of an *MT* are "request for information", "approval message", and "decline offer". Examples of an *AT* are "send request", "approve request", and "make decision".

It may seem trivial to capture events of the form *(MT,PI)* and *(AT,PI)*. In the presence of process-aware information systems such as workflow management systems (e.g., Staffware, Filenet, FLOWer, etc.) and dedicated middleware products (e.g., MQSeries and Oracle BPEL) it is indeed easy to extract the desired information. However, in many other situations this turns out to be much more complicated. In a SOA at any point in time there may be an arbitrary number of process instances all exchanging messages

[6] This requirement is also found in process mining techniques [9] (e.g., the α algorithm [11]).

using *ad hoc* means of identification, cf. the correlation mechanism of BPEL. Moreover, the same activity may appear multiple times in the process model or multiple activities may potentially send or receive similar messages.

We assume that it is possible to map messages onto events of the form *(MT,PI)*, where MT is a message type and PI is a process instance. Moreover, we assume there is a specification of the service interaction behavior in the form of an abstract BPEL process. Importantly, we do not assume that the service itself is implemented as an executable BPEL process. It may be implemented using any programming language or platform (e.g. Java). Figure 4 illustrates our assumptions: There is an abstract BPEL process and observed messages can be linked to BPEL activities such as `receive`, `invoke`, or `reply`.

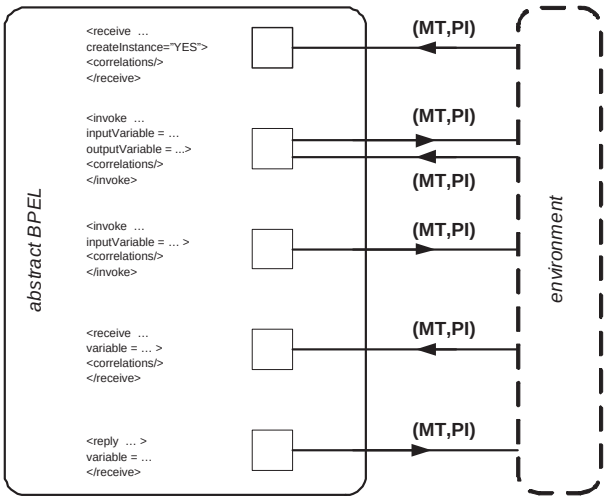


Figure 4. The situation illustrated by Figure 3(c) put into the context of BPEL with its basic activities `receive` (initial or not), `invoke` (synchronous or asynchronous), and `reply`.

In an ideal situation the abstract BPEL process and the observed messages conform (a precise definition will be given in the next section). However, there may be discrepancies between the left and the right-hand side of Figure 4. There are two possible causes for non-conformance: (1) the service implements a process different from the specification given by the abstract BPEL process; and (2) the environment behaves different from

what could be expected based on the specification given by the abstract BPEL process. In the remainder, we will show that it is indeed possible to measure conformance and track down discrepancies between the abstract BPEL process and the observed message exchanges. Although, we have implemented this in the context of BPEL, other languages for service interaction specification could be used. The only requirement is that these languages should be suitable for describing all message exchanges between the services involved in the choreography, and there should be a mapping from the control-flow subset of that language to Petri nets (and more specifically WF-nets).

As shown in Figure 2 the next three sections show how conformance checking can be applied to service behavior. First, we present an approach to do conformance checking given a Petri net and an event log (Section 3). Then we show a mapping from abstract BPEL to Petri nets (Section 4) and discuss the various ways in which service behavior (e.g., SOAP messages) can be captured and mapped onto a format suitable for conformance checking (Section 5).

3 Conformance Checking Based on Petri nets

The starting point for conformance checking is the presence of both an explicit process model, describing how some business process *should be* executed, and some kind of event log, giving insight into how it *was actually* carried out. Clearly, it is interesting to know whether they conform to each other. In [46] this question has been explored using Petri nets to represent process models [22], and assuming some abstract event log where log events are only expected to (i) refer to an activity from the business process (denoted as AT/MT in Figure 3 and 4), (ii) refer to a case (i.e., a process instance, cf. labelled PI in Figure 3 and 4), and (iii) be totally ordered.

We have identified two dimensions of conformance, *fitness* and *appropriateness* [46]. Fitness relates to the question whether the process behavior observed complies with the control flow specified by the process model, while appropriateness can be used to evaluate whether the model describes the observed process in a suitable way (cf. Occam's razor as discussed in Section 1).

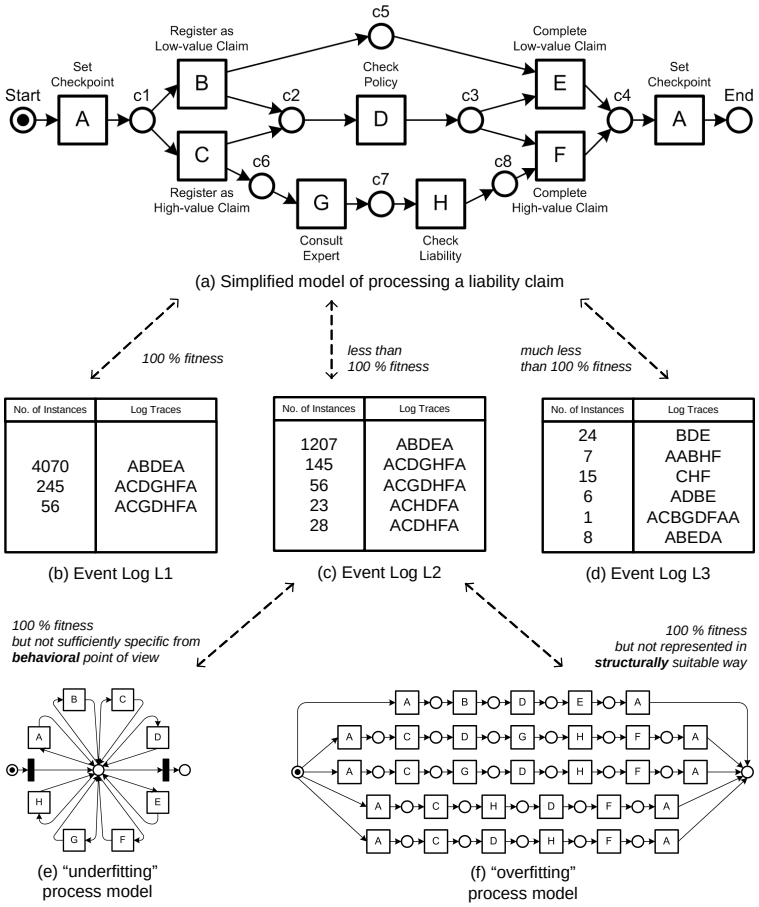


Figure 5. Two dimensions of conformance: fitness and appropriateness.

To illustrate both dimensions of conformance we use the example process shown in Figure 5(a). The process is represented as a Petri net [22]. The squares in Figure 5(a) are transitions and represent activities. The circles are places and represent pre- and post-conditions (i.e., partial states). In a Petri net, places may hold tokens. The marking of a Petri net is the distribution of tokens over places (i.e., the state). The network structure is static while the number of tokens and their location may change. A transition is enabled if there is a token on each of its input places. A transition may fire if it is enabled. Firing implies removing tokens from the input places and producing tokens for the output places. In Figure 5(a), transition A is enabled. Firing A implies moving a token from place *Start* to place *c1*, etc. There are two transitions bearing the same label "Set Checkpoint". Each of these two transitions represents an activity that 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 creating alternative paths for low-value claims and high-value claims, i.e., claims get registered differently (B or C) depending on their value. 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). Finally, the claim is completed according to the former choice between B and C (i.e., E or F).

Figures 5(b)-(d) show three example logs for the process described in Figure 5(a) at an aggregate level. This means that process instances exhibiting the same event sequence are combined as a logical log trace while recording the number of instances to weigh the importance of that trace. That is possible since 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.

Event log *L1* completely *fits* the model in Figure 5(a) as every log trace can be associated with a valid path from *Start* to *End*. In contrast, event log *L2* does not match completely as the traces *ACHDFA* and *ACDHFA* lack the execution of activity G, while event log *L3* does not contain any trace corresponding to the specified behavior.

Now consider the two process models shown in Figure 5(e)-(f). Although event log *L2* fits both models quantitatively, 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 observed behavior. The first one is much too generic ("underfitting") as it covers a lot of extra behavior, allowing for arbitrary sequences containing the activities A, B, C, D, E, F, G, or H, while the latter—although it does not allow for more sequences than those that were observed in the log—only lists the possible behavior instead of expressing it in a meaningful way ("overfitting"). Note that such underfitting and overfitting models could be constructed for any log, e.g., also *L1* and *L3* in Figure 5. Therefore, these extremes do

not offer a better understanding than can be obtained by just looking at the aggregated log. So, there is also a qualitative dimension and we claim that a "good" process model should somehow be minimal in structure to clearly reflect the described behavior (i.e., *structural appropriateness*), and minimal in behavior to represent as closely as possible what actually takes place (i.e., *behavioral appropriateness*).

Conformance checking aims at both quantifying the respective dimension of conformance and locating the mismatch, if any. Therefore, we have developed metrics for measuring the fitness, and the behavioral and structural appropriateness of a given process model and event log [46]. But we also seek for suitable visualizations of the results as this is crucial for giving useful feedback to the analyst.

For example, we can quantify fitness by replaying the log in the model. For this, the replay of every log trace starts with marking the initial place in the model and then the transitions that belong to the logged events in the trace are fired one after another. While doing so, one counts 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 (they indicate that the process has not *properly completed*). Only if there were neither tokens left nor missing, the fitness measure evaluates to 1.0, which indicates 100 % fitness.

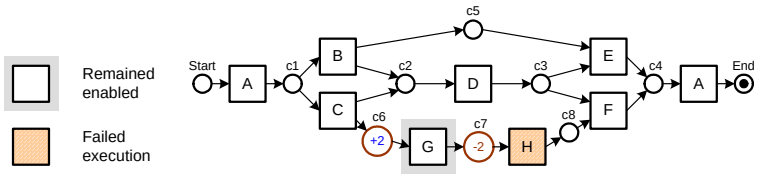


Figure 6. Example process model after replay of event log *L2*.

Figure 6 shows that the places of missing and remaining tokens during log replay can also be used to provide insight into the location of error. Because of the remaining tokens (whose amount is indicated by a + sign) in place *c6* transition *G* has remained

enabled, and as there were tokens missing (indicated by a $-$ sign) in place $c7$ transition H has failed seamless execution. This suggests that the expert consultation (activity G) did not take place for all the treated cases, and possible alignment actions would be to either enforce the specified process or to introduce the possibility to skip activity G in the model.

Both dimensions of conformance, i.e., fitness and appropriateness, have been implemented in the ProM Conformance Checker [46]. Note that the checker supports *duplicate activities*, e.g., in Figure 5(a) there are two activities with label A. This is important because multiple activities in a BPEL specification can exchange messages of a given type and are therefore indistinguishable. Similarly, it is important that the Conformance Checker supports *silent steps*, i.e., activities that are not logged. Note that the presence of silent activities makes it necessary to construct parts of the state space to find the most likely path.

4 Mapping BPEL onto WF-nets

To provide tool support for conformance checking of BPEL processes we rely on two tools developed by the authors of this paper: BPEL2PNML and WofBPEL. BPEL2PNML translates BPEL process definitions into Petri nets represented in the Petri Net Markup Language (PNML). WofBPEL, built using Woflan [51], applies static analysis and transformation techniques on the output produced by BPEL2PNML. For the purpose of conformance checking, WofBPEL is used to: (i) simplify the Petri net produced by BPEL2PNML by removing unnecessary silent transitions, and (ii) convert the Petri net into a so-called WorkFlow net (WF-net), which has certain properties that simplify the analysis phase and is the input format required by the ProM Conformance Checker.

Below, we discuss the mapping from BPEL to WF-nets and illustrate it using a BPEL process definition of a supplier service that we will use as a running example in the remainder of this paper.

4.1 The Supplier Service

Figure 7 provides an overview of a "Supplier Service" using a visual notation reflecting the syntax of BPEL. This service provides a purchase order and change order service for customers, where the purchase order that has been placed may be changed once.

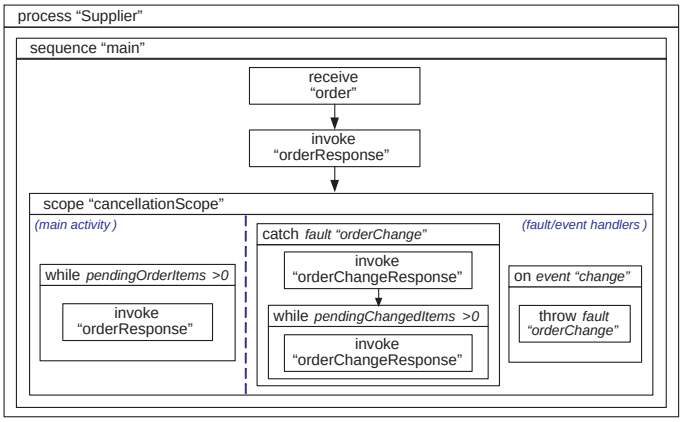


Figure 7. An abstract view of the Supplier process.

The Supplier process is initiated upon receiving a purchase order that contains one or several line items. The supplier may accept or reject any ordered item, possibly suggesting alternative products, quantities or delivery dates in the latter case. The supplier replies to the purchase order either with a single response listing the outcome for all items, or with multiple responses corresponding to subsets of the items. The rationale for having multiple responses is that the supplier may be unable to determine outright if it can accept a line item. In this case, the supplier sends a first response listing the items of which the outcomes have been determined. Additional responses are then sent as information becomes available. After receiving an order response, the customer may request to change the previous purchase order because of some item(s) being rejected. A change order is an updated purchase order that overrides the previous one. Similarly to the processing of a purchase order, the supplier may reply with a single response or with multiple responses to a change order.

For the readers that want to see the actual definitions of the Supplier service both as an abstract and as an executable BPEL process, we refer to a technical report where we include the full BPEL specifications [4].[7] An abstract process is defined at the level of abstraction required to capture public aspects of the service (i.e., message exchanges with the environment). In the working example, the abstract process specifies that the service receives orders and change orders and sends order responses and change order responses, and captures the control dependencies between these messages. Meanwhile, an executable process represents a possible implementation of the abstract process. However, services are not always coded as BPEL executable processes. Not untypically, services are coded in mainstream programming languages (e.g., Java). If the Supplier service is implemented as a BPEL executable process, it is possible to collect logs of the form (AT, PI) in the terminology of Figure 3. This enables conformance checking based on the activity-oriented logs as illustrated by figures 3(b) and 3(d). Otherwise, conformance checking in the style of figures 3(a) and 3(c) (i.e., based on messages) can be performed by comparing a BPEL abstract process describing the expected behavior of the Supplier service with actual message logs of the form (MT, PI).

4.2 Mapping BPEL to Petri Nets

We first map BPEL processes to Petri nets, which can be then converted to WF-nets. When using Petri nets to capture the formal semantics of BPEL, we allow the usage of both labeled and unlabeled transitions. The labeled transitions model events and basic activities. The unlabeled transitions (τ-transitions, also known as *silent steps*) represent internal actions that cannot be observed by external users. This section presents only selected parts of the mapping. A complete version of the formal specification of the mapping can be found in [44].

Activities We start with the mapping of a basic activity (X) shown in Figure 8, which also illustrates our mapping approach for structured activities. The net is divided into

[7] This report can be downloaded from `http://BPMcenter.org`.

two parts: one (drawn in solid lines) models the normal processing of X, the other (drawn using dashed lines) models the skipping of X.

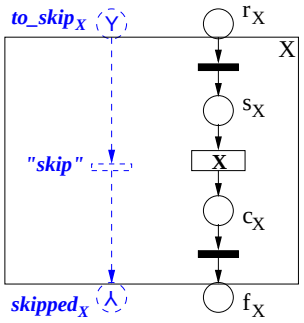


Figure 8. A Basic activity.

In the normal processing part, the four places are used to capture four possible states for the execution of activity X: $\mathsf{r_X}$ for "ready" state, $\mathsf{s_X}$ for "started" state, $\mathsf{c_X}$ for "completed" state, and $\mathsf{f_X}$ for "finished" state. The transition labeled X models the action to be performed. This is an abstract way of modeling basic activities, where the core of each activity is considered as an atomic action. Two τ-transitions (drawn as solid bars) model silent steps, i.e., internal actions for checking pre-conditions or evaluating post-conditions for activities. In the mapping of BPEL to Petri nets, we will introduce many silent steps to model the "logical wiring" among transitions representing the actual activities. The skip path in Figure 8 is mainly used to facilitate the mapping of control links. Note that the $\mathsf{to\_skip}$ and $\mathsf{skipped}$ places are respectively decorated by two patterns (a letter Y and its upside-down image) so that they can be graphically identified. In Figure 8, hiding the subnet enclosed in the box labeled X yields an abstract graphic representation of the mapping for activities. This is used in the rest of the paper.

Figure 9 depicts the mapping of structured activities. Next to the mapping of each activity is a BPEL snippet of the activity. More τ-transitions (drawn as hollow bars) are introduced for the mapping of routing constructs. In Figure 9 and subsequent figures, the skip path of the mapping is not shown if it is not used. A detailed description of the

mapping [44] is outside the scope of this paper. However, to give some insight into the mapping, we describe the mappings of *while* and *scope* activities in some detail.

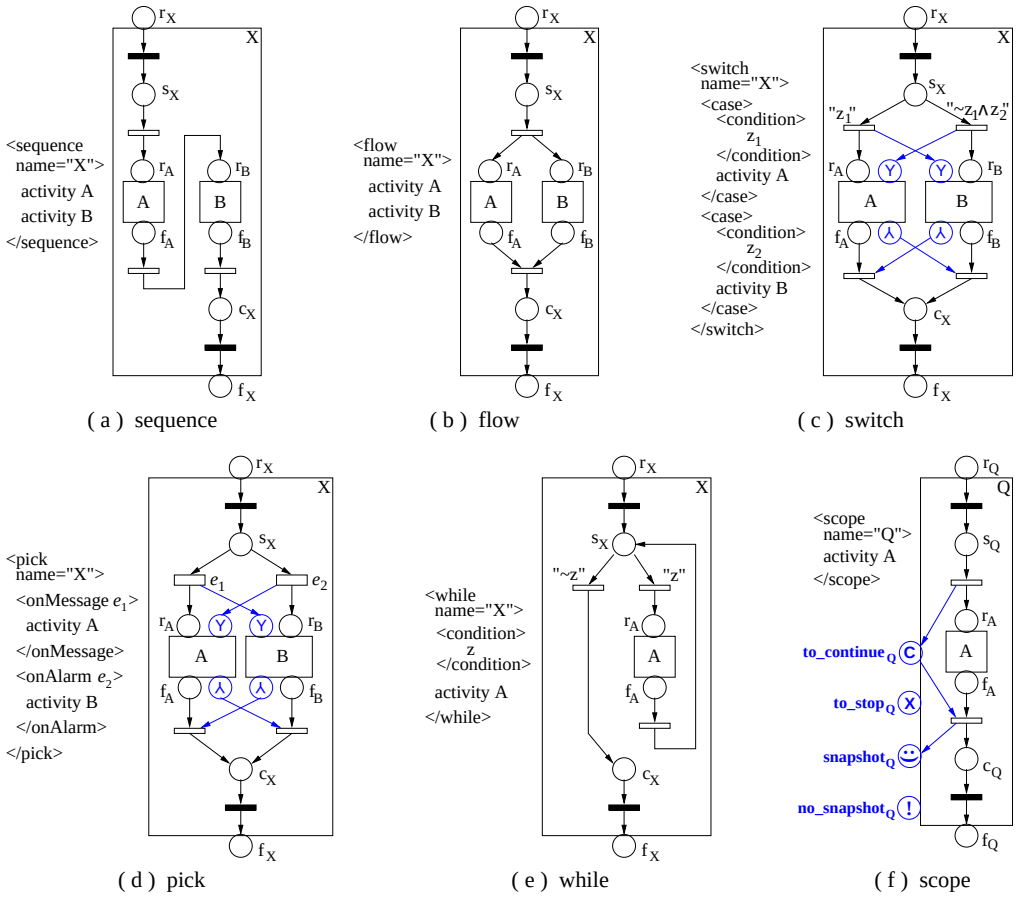


Figure 9. Mapping structured activities.

A *while* activity supports structured loops. In Figure 9(e), activity X has a sub-activity A that is performed multiple times as long as the while condition (z) holds and the loop construct ends if the condition does not hold anymore (∼z).

A *scope* provides event and fault handling. It has a main activity that defines its "normal" behavior. To map fault handling, we define four flags for a scope, each represented by a Petri net place, as shown in Figure 9(f). These flags are: to_continue, indicating the execution of the scope is in progress and no exception has occurred; to_stop, signaling an error has occurred and all active activities nested in the scope need to stop; snapshot,

capturing the *scope snapshot* defined in [14] which refers to the preserved state of a successfully completed uncompensated scope; and no_snapshot, indicating the absence of a scope snapshot. Specifically, if a fault occurs during the execution of the normal behavior associated to a scope, it will be caught by one of the fault handlers defined for the scope, and the scope switches from normal "processing" mode to "fault handling" mode. These two modes are represented by places to_continue and to_stop. A scope in which a fault has occurred is considered to have ended abnormally and thus cannot be compensated, even if the fault has been caught and handled successfully. This is represented by places snapshot and no_snapshot. For space reasons, we do not describe fault handlers and other advanced constructs. Full details, including a formal definition of the mapping, can be found in a separate technical report [44].

Event Handlers A scope can provide *event handlers* that are responsible for handling normal events (i.e., message or alarm events) that occur *concurrently* when the scope is running. Figure 10 depicts the mapping of a scope (Q) with an event handler (EH). The four flags associated with the scope are omitted. The subnet enclosed in the box labeled EH specifies the mapping of EH. As soon as scope Q starts, it is ready *to invoke* EH. Event $\mathsf{e}_{\mathsf{normal}}$ is *enabled* and may occur upon an environment or a system trigger. When $\mathsf{e}_{\mathsf{normal}}$ occurs, an instance of EH is created, in which activity HE ("handling event") is executed. EH remains active as long as Q is active. Finally, event $\mathsf{e}_{\mathsf{normal}}$ becomes disabled once the normal process (i.e., main activity A) of Q is finished. However, if a new instance of EH has already started before $\mathsf{e}_{\mathsf{normal}}$ is disabled, it is allowed to complete. The completion of the scope as a whole is delayed until all active instances of event handlers have completed.

Example: Mapping of the Supplier Process Figure 11 depicts the mapping of the Supplier process shown in Figure 7. The complete mapping of the Supplier process, as obtained using BPEL2PNML, is summarized in Figure 11. This figure sketches the top-level structure including the top-level scope, the fault handler and the event handler

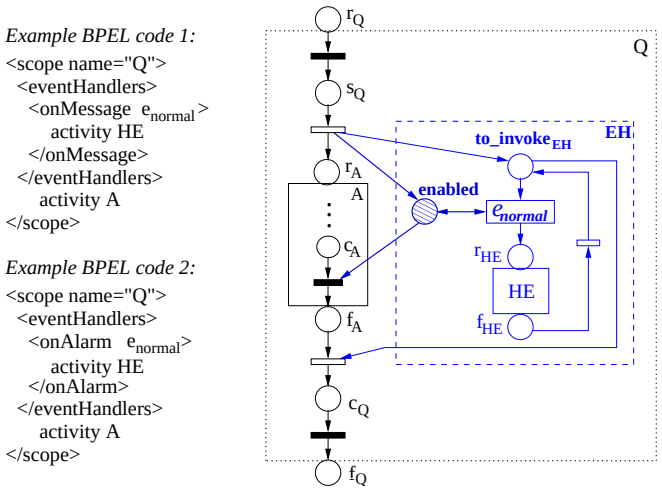
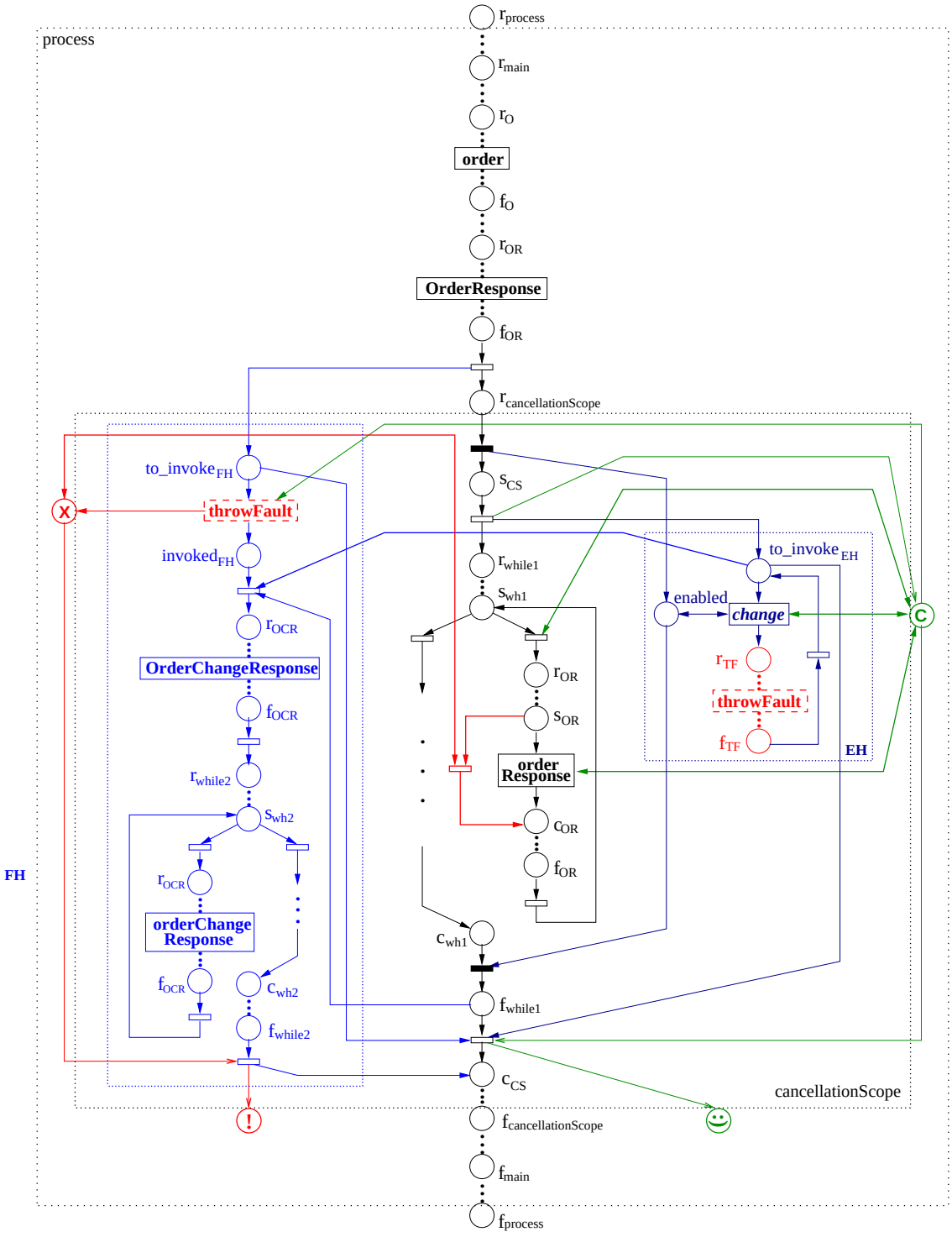


Figure 10. Mapping event handlers.

in the process. For illustration purposes, some net details (e.g., those associated to the process scope and the skip paths) are omitted. Also, for the sake of readability the following conventions are used: place to_continue is labeled by a "C", to_stop by an "X", snapshot by a "smiley face" and no_snapshot by an exclamation mark. The reader does not need to understand this diagram in detail. What is important to retain is that any abstract BPEL specification can be mapped onto a Petri net, but in this mapping process a large number of silent steps (i.e., transitions with label τ) are introduced. Next we will show how to remove these and simplify the Petri net prior to performing conformance checking.

4.3 From Petri nets to WF-nets

The ProM Conformance Checker takes a WF-net [1] and an MXML log [24] as input. A WF-net is a Petri net which models a workflow process definition. It has exactly one input place (called source place) and one output place (sink place). A token in the source place corresponds to a case (i.e., process instance) which needs to be handled, and a token in the sink place corresponds to a case which has been handled. Also, in a WF-net there are no dangling tasks and/or conditions. Tasks are modeled by transitions and conditions by places. Therefore, every transition/place should be located on a path from the source place to the sink place in a WF-net [1].



[Note]: The concrete action of ''throwFault'' is modelled by one transition, which is graphically represented by two transitions throwFault to avoid arc crossing.

Figure 11. Mapping of the Supplier process shown in Figure 7.

The Petri net obtained from the automated mapping to Petri nets is generally not a WF-net. For example, the Petri net sketched in Figure 11 contains four sink places: one at the bottom of the figure and four along the dotted line labelled "cancellationScope". Such additional source and sink places come from the mapping of constructs that can cause activities to be skipped, namely: control links and fault handlers attached to scopes. In order to facilitate the mapping of control links and fault handlers, and to be consistent in the way structured activities are mapped in BPEL, we have assumed in our mapping that any activity may be skipped. As a result, a skip path is generated for every activity in BPEL2PNML. However, not every activity can actually be skipped. A straightforward counter example is the root activity (i.e., the top-level process scope). By removing these idle skip fragments, the Petri net obtained from the initial phase of the mapping can be converted to a WF-net.

We use WofBPEL to convert the Petri nets returned by BPEL2PNML to WF-nets. WofBPEL has originally been built to perform analysis on the Petri nets produced as output from BPEL2PNML. Since it uses Woflan [51] and Woflan can only handle WF-nets, WofBPEL first need to remove the idle skip fragments to obtain a WF-Net. In addition to this, WofBPEL also applies behavior preserving reduction rules based on the ones given by Murata [42]. This way, the size of the net can be significantly reduced by removing unnecessary silent transitions and redundant places. Note that there is a difference between the rules given by Murata and the rules used in WofBPEL. The explanation for this difference is that in our case the non-silent transitions (represented by labeled transitions) should never be removed.

Figure 12 visualizes the reduction rules used in WofBPEL, where only silent transitions (τ-transitions) can be removed. The first rule shows that a (silent) transition connecting two places may be removed by merging the two places, provided that tokens in the first place can only move to the second place. The second rule shows that multiple alterative silent transitions can be reduced to a single one. Note that after applying the second rule one may be able to apply the first rule provided that the first place has only

one remaining output arc (see Figure 12). The third rule shows that self-loops can be removed if the transition involved is silent. When applying the rules one should clearly differentiate between silent and non-silent transitions. For example, in the fourth and fifth rule at least one of the transitions should be silent, otherwise the rule should not be applied (as indicated). In the fourth rule the execution of y is inevitable once the silent transition has been executed. Therefore, it is only possible to postpone its occurrence. In the fifth rule the execution of x is always followed by the silent transition. Here it is important to note that the silent transition cannot have any additional inputs. Therefore, it is only possible to postpone its occurrence. The two last rules do not remove any transitions but remove places. Note that X and Y may be or may not be silent. The reduction rules shown in Figure 12 do not preserve the moment of choice and therefore assume trace semantics rather than branching/weak bisimulation [29].

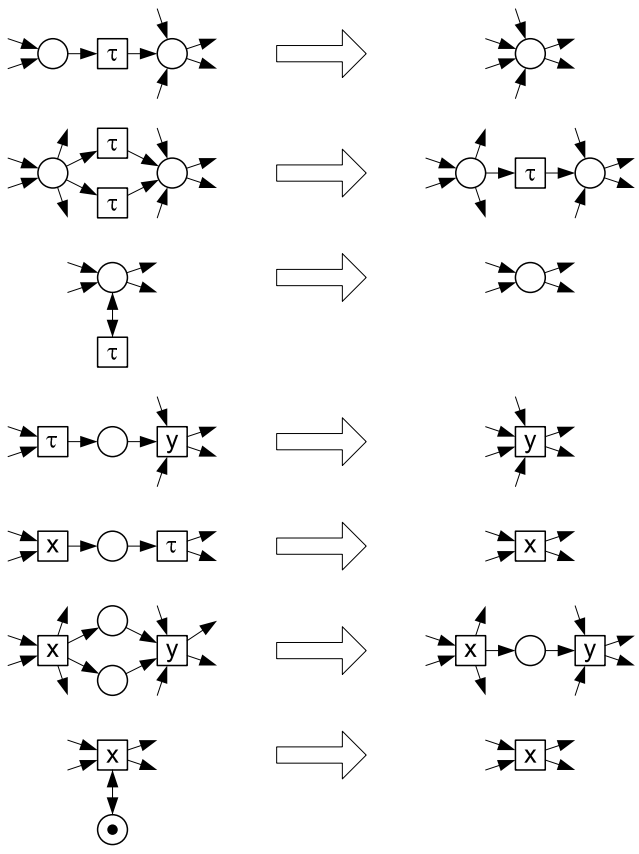


Figure 12. Behavior preserving reduction rules used in WofBPEL.

Based on the above, Figure 13 depicts the WF-net automatically generated from the Supplier BPEL process of Figure 7 using first BPEL2PNML to obtain a Petri net and then WofBPEL to remove idle skip paths and reduce the size of the net. For reference, the Petri net generated by BPEL2PNML for the Supplier process contains 96 places and 84 transitions, while the reduced WF-net contains 27 places and 27 transitions.

5 Monitoring and Correlating Messages

In order to perform conformance checking, we assume that messages sent and received by a service are logged. The resulting logs should be ordered chronologically and should contain for each message, an indication of whether the message is inbound or outbound, as well as the message headers (e.g., HTTP and/or SOAP headers). The message payload is not relevant as we focus on behavioral rather than structural conformance.

Given such a message log and a BPEL abstract process definition that is presupposed to correspond to the message log, we need to extract log traces such as those depicted in Figures 5(b)-(d).[8] The labels in these log traces should correspond to labels in the Petri net obtained from the BPEL abstract process definition. As suggested in Figure 3, these labels must allow one to determine the direction of messages (indicated by arrows in Figure 3) and its message type (designated as MT in Figure 3). Thus, for each message we must determine:

- Its corresponding BPEL abstract process instance (herewith called its *service instance*). This is required because the event log needs to be structured as a set of log traces, each one corresponding to one execution of the process capturing the expected behavior of the service.
- A label denoting the BPEL communication action in the abstract process definition to which the production or consumption of the message is attributed.

In the remainder of this section we discuss both issues in detail.

[8] Note that we will extract more information but this is the bare minimum for conformance checking. The MXML format also allows for the logging of timestamps, data, resources, and transactional aspects.

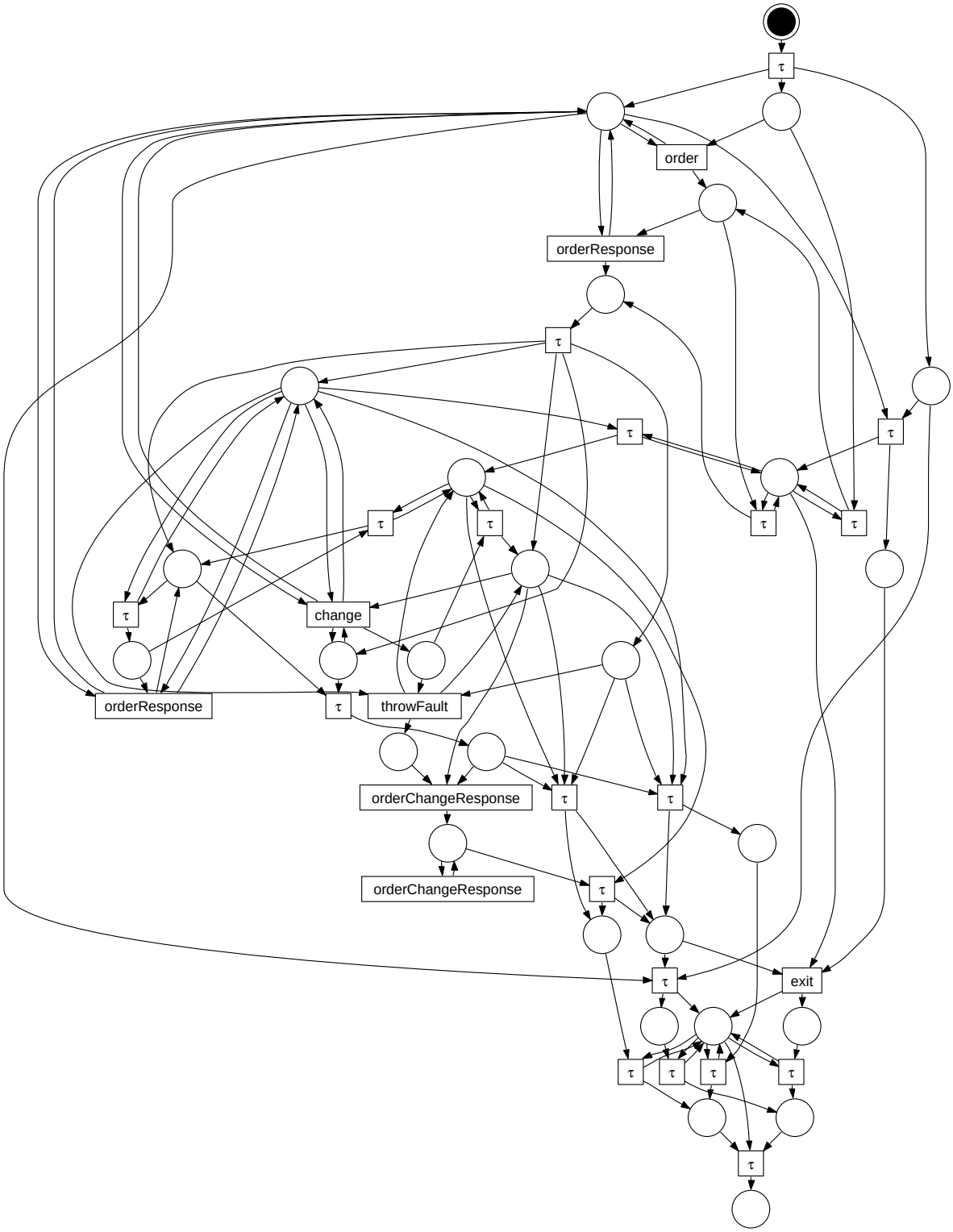


Figure 13. The WF-net for the Supplier process.

5.1 Grouping messages into log traces

In order to apply the proposed conformance checking technique, messages need to be grouped into *log traces* each representing one execution of the service, i.e., each message needs to be associated to a process instance. If the service is implemented as an executable BPEL process, this grouping of messages is trivial. The process is executed by an engine that generates logs associating each communication action (and thus the message consumed or produced by that action) to a process instance. All messages consumed or produced by a process instance can then be grouped into a log trace.

If no executable BPEL process is available, we need to group messages into log traces just by looking at their contents. Current web service standards do not make a provision for messages to include a "service instance identifier", so assuming the existence of such identifier may be unrealistic in some situations. Other monitoring approaches in the field of web services have recognized this problem and have addressed it in different ways, but they usually end up relying on very specific and sometimes proprietary approaches. For example the Web Services Navigator [45] uses IBM's Data Collector to log both the contents and context of SOAP messages. But to capture enable correlation, the Data Collector inserts a proprietary SOAP header element into messages.

In order to avoid relying on proprietary SOAP extensions, we use a generic grouping mechanism that we term *chained correlation*. The idea of chained correlation is that every message, except for the first message of a service instance, refers to at least one previous message belonging to the same service instance. In the context of contemporary web service standards and middleware this correlation information can be obtained in at least two ways:

- When using SOAP in conjunction with WS-Addressing, each message contains an identifier (*messageID* header) and may refer to a previous message through the *relatesTo* header. If we assume that these addressing headers are used to relate messages belonging to the same service instance in a chained manner, it becomes possible to group a raw service log containing all the messages sent or received by a

service into log traces corresponding to service instances. This is the method used in our case study and more details will be given in Section 6. The method is applicable when using Oracle BPEL as well as various other web service middleware supporting the WS-Addressing standard. Note however web service middleware supporting WS-Addressing may use the *replyTo* header to correlate messages as opposed to the *relatesTo*. Specifically, the *replyTo* header of a given message (say M) may contain a URI uniquely identifying the message in question. Subsequently, when another message M' of the opposite directionality is observed that has the same URI this time in the *To* header, M and M' can be correlated.

– The second method is based on the identification of properties that a message has in common with another message belonging to the same service instance. In BPEL, properties shared by messages belonging to the same service instance are captured as *correlation sets*. A correlation set can be seen as a function that maps a message to a value of some type. Correlation sets are associated with communication actions. When a message is received which has the same value for a correlation set as the value of a message previously sent by a running service instance, the message in question is associated with this instance. This allows one to map messages to service instances, except for those messages that initialize a correlation set, that is, those messages that start a new instance. Assuming that in the BPEL abstract process of a service only the initial actions of the protocol initialize correlation sets, and all other actions refer to the same correlation sets as the initial action, each message produced or consumed by the service can be mapped to a service instance as follows: The full message log is scanned in chronological order. A message is either related to a new service instance if it corresponds to a communication action that initializes a correlation set, or related to a previously identified service instance if the values of its correlation set match those of a message sent by the previous service instance. This method can be applied in the scenario of Figure 3(a) where the message logs and an abstract BPEL process are available.

In some cases, neither of the techniques outlined above is applicable. In other words, there may be no way of defining a function that can determine whether or not a given message is related to a previously observed message. In this case, techniques from the area of Web session identification can be employed, but such techniques are not 100% reliable. This avenue is considered in [30].

5.2 Abstracting messages as labels

Once the message log has been grouped into log traces corresponding to service instances, we associate each message in a log trace with a transition label used in the WF-net obtained from the BPEL abstract process definition. These transition labels represent communication actions seen at the level of abstraction used for conformance checking.

BPEL's communication action types are: *invoke*, *reply*, *receive*, and *onMessage* (or *onEvent* in BPEL 2.0). A receive or an onMessage action consumes one message, a reply produces one message, while an invoke can either produce a single message (*simple send*) or produce a message and consume another one in that order (*synchronous send-receive*). Without loss of generality, we assume that the BPEL abstract process given to the Conformance Checker does not contain any synchronous send-receive. For the purposes of conformance checking, a synchronous send-receive can be decomposed into a *sequence* activity containing a simple send followed by a receive. Also without loss of generality, we assimilate reply actions to send actions and onMessage handlers to receive actions, since these elements have the same effect in terms of message logs.

Thus, for conformance checking purposes, we view communication actions in a BPEL abstract process as being labeled by a pair $\langle$ D, MT $\rangle$ where D stands for the direction (inbound or outbound) and MT for message type as in Figure 3. All non-communication actions are given τ-labels since their execution does not manifest itself as message log entries. Actions with τ-labels in the abstract process get translated to silent transitions.

Under this labeling scheme, it is possible that two actions in a BPEL process get the same label. Hence, the Petri net generated from a BPEL abstract process may have multiple (non-silent) transitions with the same label. Fortunately, this possibility

is supported by the conformance checking technique, e.g., the example in Figure 5(a) contains two actions with label A.

Each communication action in a BPEL process definition is linked to a WSDL operation. A WSDL operation in turn is associated with *binding information* that determines how messages related to that operation are encoded and exchanged over a given communication protocol (e.g., SOAP over HTTP or XML over HTTP). The structure of an operation's binding information varies depending on the transport protocol, but in any case it normally provides a means to identify messages that pertain to that operation. In the case of SOAP over HTTP, the binding information for a WSDL operation maps this operation to a *SOAPAction* identifier. This makes it possible to reliably associate a SOAP message with a WSDL operation by inspecting the *SOAPAction* field in the HTTP header of the message. In the case of a communication protocol based on plain XML over HTTP, the binding information of a given WSDL operation may include a relative URL to be found in the HTTP headers of every message pertaining to that operation. Again, this makes it possible to associate a SOAP message to an operation by analyzing the "request URI" in the message's HTTP header.

In the general case however, the SOAPAction header and the mapping between WSDL operations and SOAPAction identifiers are optional. In the absence of this information, associating SOAP messages to WSDL operation may require inspection of the message's body. Specificially, the top-level element in the SOAP message body needs to be compared with the message type associated to each operation supported by the service. This technique is only reliable if operations map to message types with different top-level elements. Otherwise, the user of the conformance checker would have to provide a function mapping each SOAP message in the log to a WSDL operation. This illustrates that the versatility of SOAP and WSDL make it difficult to achieve a general and reliable solution to the problem of mapping messages to operations. In some cases, tailor-made solutions are required.

Having analyzed this problem and outlined possible approaches, we assume that every message produced or consumed by a service for which a BPEL abstract process is defined can be mapped to a WSDL operation. With this information and the message direction, it is possible to construct log traces such that each entry in the trace can be matched to a communication action label under the labeling scheme described above. Because we are able to map a BPEL specification onto a Petri net (cf. Section 4) and we can associate messages to both process instances and activities (cf. this section), we can apply the conformance checking techniques described in Section 3.

6 Example

In this section we apply our findings to the example BPEL process introduced in Section 4. The goal of this section is to demonstrate the applicability of our approach and tools (BPEL2PNML, WofBPEL, and the ProM Conformance Checker). To generate SOAP messages we need to implement the process specified in terms of abstract BPEL. We could have used a conventional language to do this. However, we chose not to do so and implemented the executable Supplier process using Oracle BPEL 10.1.2, i.e., the current BPEL offering of Oracle. After implementing the executable BPEL process, we obtained different logs by monitoring the SOAP messages between two Oracle BPEL servers (one for the supplier and one for the customer). Note that we use the local message observer setting for choreography conformance checking as shown in Figure 3(c). Using these logs we will show that we can correlate the SOAP messages that belong to the same process instance (as discussed in Section 5). Finally, we show that we can successfully check the conformance of the abstract Supplier BPEL process using the techniques described in Section 3.

6.1 Executable Supplier BPEL process

We have implemented the executable Supplier BPEL process in Oracle BPEL Process Manager 10.1.2. Figure 14 shows the process using JDeveloper. On the far left, we see

a number of partner links: TaskManagerService, TaskActionHandler, TaskRoutingService, IdentityService, and client. The first four partner links are used for the user tasks handling the purchase order (or change order), the remaining client service is used for the customer.

The pane in the middle contains the actual process, which consists of three 'threads'. The leftmost thread handles a purchase order and its responses. The rightmost thread handles the receipt of a change order: It generates a fault which preempts the leftmost thread and starts the middle thread. The middle thread handles the responses to the change order.

After having deployed this process, we can initiate it from Oracle's BPEL Console. Using Oracle's BPEL Worklist, we can handle the purchase order and send responses back to the customer. A change order can be initiated from the BPEL Console, and the corresponding change order can be handled using the BPEL Worklist. As an example, Figure 15 shows a work item corresponding to the purchase order when two (out of five) line items are about to be accepted.

6.2 SOAP messages

SOAP messages are typically sent between two processes, which can both run on the same server or on different servers. For this reason, we also implemented a simple Customer executable BPEL process. The Customer process places an order, waits for an orderResponse, then places a changeOrder, waits for two orderChangeResponses, and then exits. We ran the processes on two different (Oracle BPEL 10.1.2) servers.

Unfortunately, we were unable to obtain the SOAP messages directly from Oracle BPEL. No option existed to log all SOAP messages send and/or received to some file, and they were also not stored in the database underlying the Oracle BPEL server. As a result, we had to use a TCP Tunneling technique[9] to obtain the SOAP messages. With this technique, it is fairly easy to eavesdrop on a specific combination of host and port. Typically, incoming messages all go to the same combination of host and port, but

[9] see http://www.oracle.com/technology/products/ias/bpel/htdocs/orabpel_technotes.tn001.html

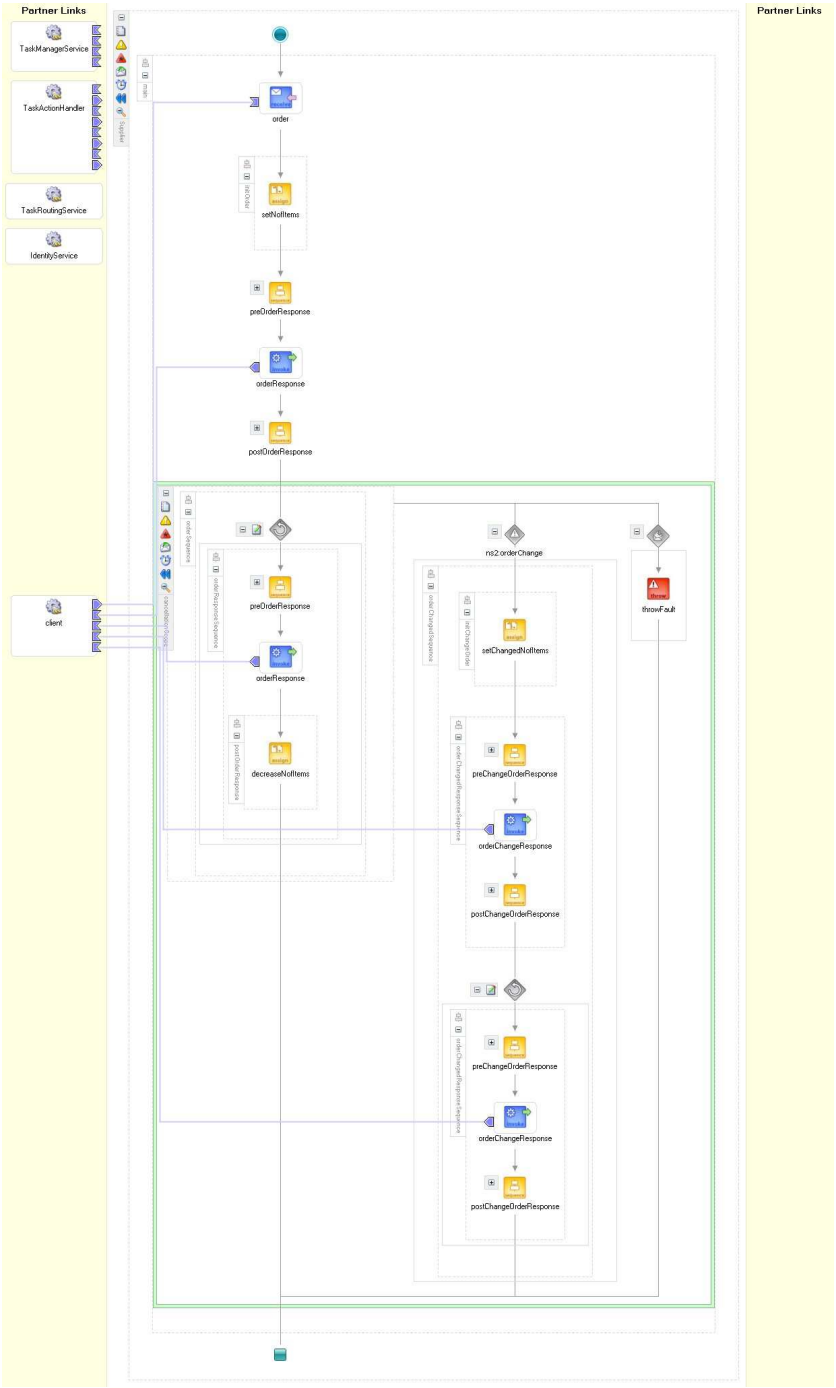


Figure 14. The Supplier process using JDeveloper.

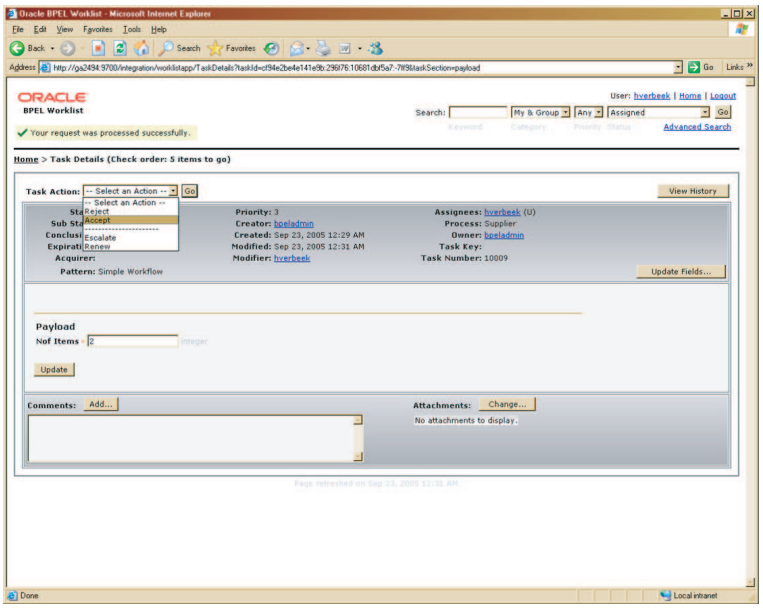


Figure 15. A work item in Oracle BPEL Worklist.

outgoing messages can be directed to a multitude of combinations of hosts and ports. As a result, it is more convenient to eavesdrop on the incoming messages.

Examples of the SOAP message logs from both servers are given in the appendix of our technical report [4].

6.3 Message correlation

From the SOAP message logs, it is straightforward to generate a log as shown in Figure 16: The first message (the order) contains a unique message id (`bpel://localhost/default/Customer~1.1/301-BpInv0-BpSeq0.3-3`), and all other related messages relate to this message id.

Both the WF-net corresponding to the abstract Supplier process (cf. Figure 13) and the log from Figure 16 can be imported by the ProM framework to check their conformance.[10]

```
<?xml version="1.0" encoding="UTF-8"?>
<WorkflowLog>
  <Source
    program="Oracle BPEL, using TCP Tunneling"
  />
  <Process
    id="http://services.qut.com/Supplier"
    description="Supplier 1.1, using Customer 1.1 as customer stub"
  >
    <ProcessInstance
        id="bpel://localhost/default/Customer~1.1/301-BpInv0-BpSeq0.3-3"
        description="Instance 301"
    >
      <AuditTrailEntry>
        <WorkflowModelElement>order</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>2005-10-20T11:54:09-00:00</Timestamp>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>orderResponse</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>2005-10-20T11:58:08-00:00</Timestamp>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>change</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>2005-10-20T11:58:20-00:00</Timestamp>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>orderChangeResponse</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>2005-10-20T11:58:35-00:00</Timestamp>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>orderChangeResponse</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>2005-10-20T11:58:43-00:00</Timestamp>
      </AuditTrailEntry>
    </ProcessInstance>
  </Process>
</WorkflowLog>
```

Figure 16. An example SOAP-based log.

Table 1. Desirable and undesirable scenarios for the supplier service execution.

	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)

6.4 Conformance checking

Having demonstrated that it is feasible to obtain an event log (such as in Figure 16) from real service executions, we now use conformance checking techniques (see also Section 3) to validate the supplier service specification for a number of interaction scenarios. Table 1 shows five execution sequences which should be valid for the supplier service as specified in Section 4.1 and eight which should not.

Scenarios 1 – 5 reflect message sequences which should be compliant with the process specification (note that Scenario 5 corresponds to the example from Figure 16). They all start with an initiating *order*, followed by one or more *orderResponses*, and potentially complete with a *change* request and one or more *orderChangeResponses*.

Scenarios 6 – 13 represent conceivable settings of misbehavior, whereas 6 – 9 correspond to possible violations by the supplier service and 10 – 13 contain violations by the client or environment of the service. Both Scenario 6 and 7 show situations where the conversation has not been completed properly as after having received the *order* request the service needs to send at least one *orderResponse* (missing in Scenario 6), and following a *change* request at least one *orderChangeResponse* must be sent (missing in Scenario 7). In Scenario 8 the supplier service sends an *orderResponse* which is not

[10] Both the corresponding schema definition and the ProMimport framework, which converts logs from existing (commercial) PAIS to the XML format used by ProM, can be downloaded from `www.processmining.org`.

correlated with a previous *order*, and in Scenario 9 it still sends another *orderResponse* although a *change* request has been received already (and thus only *orderChangeResponses* should be sent). Scenario 10 shows the situation where the environment invokes a *change* request although the first *orderResponse* has not been sent by the service yet. In Scenario 11 a *change* request is invoked which is not even related to a previous *order*. Both Scenario 12 and 13 show a situation in which a second *change* is requested by the client, which is not allowed.

In order to verify the given scenarios with respect to the supplier service specification from Section 4.1 we use the reduced Petri net model generated from the abstract BPEL process, shown in Figure 13. Having imported it into the ProM framework, the Conformance Checker [46] is able to replay the log containing the scenarios in the model. Based on the number of missing and remaining tokens the fitness measurement is calculated indicating whether a scenario corresponds to a valid execution sequence for that process. If not, the depiction of missing and remaining tokens aids in locating the problem.

Consider for example Figure 17, in which the Conformance Checker shows a part of the model after the replay of Scenario 8. In this situation a single *orderResponse* is sent without having received any previous *order*. The place in the upper left corner which has no incoming arcs represents the start place of the whole process (i.e., a token will be put there in order to start the replay of the scenario). Following the control flow of the model it can be observed that the *order* transition is supposed to fire first in order 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

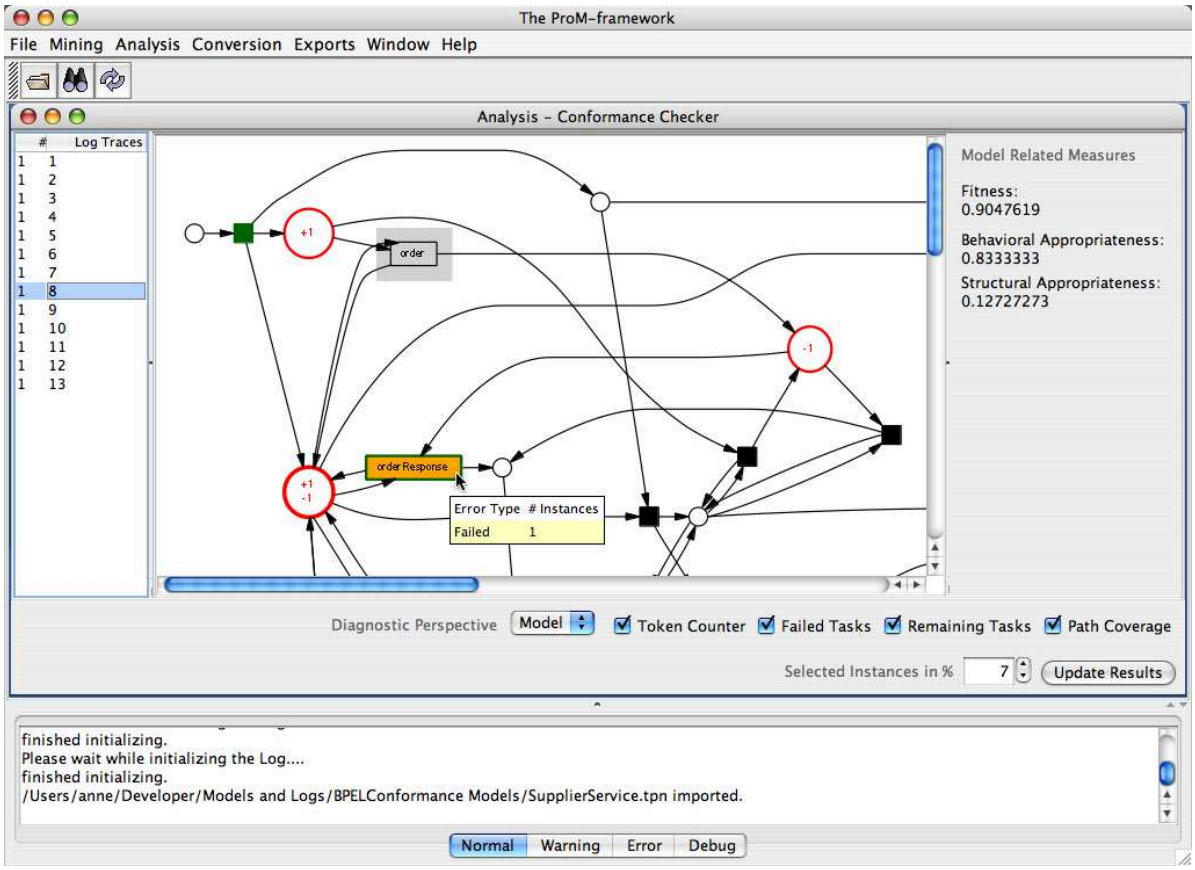


Figure 17. The Conformance Checker analyzing the scenarios from Table 1.

lower left corner (indicated by a + sign), which leads to the *order* transition remaining enabled after replay has finished. Remaining tasks are depicted with a shaded rectangle in the background and they hint that their execution would have been expected.

Now reconsider Table 1 where the Fitness column indicates for each scenario whether it corresponds to a valid execution sequence for our supplier service (i.e., during replay there were neither tokens missing nor remaining and therefore fitness = 1.0) or not (i.e., fitness < 1.0). As it shows 100 % fitness for Scenario 1 – 5 we have proven the abstract BPEL process being a valid specification with respect to the "well-behaving" conversation scenarios we thought of. However, it also allows for an execution sequence that we have classified as undesirable behavior, namely Scenario 9: Although another *orderResponse* is sent after a *change* request has been received already (and thus only *orderChangeResponses* should be sent) the scenario turns out to comply with the given

abstract BPEL process specification. This is an interesting result as it makes us aware of the fact that—due to a number of intermediate states—the chosen fault/event handler construct does not completely capture the intended constraint.

As mentioned in Section 3, there is another dimension of conformance besides fitness we are interested in: *appropriateness*. Appropriateness relates to the question whether the model is a suitable representation for the process that has been observed in the log. The Conformance Checker supports both a metric for structural and for behavioral appropriateness (the definitions and further details are provided in [46]). As for the structural appropriateness, the reduced Petri net depicted in Figure 13 has been measured to be 0.127, which is a relatively low value caused by the many silent transitions (τ-transitions). However, measuring the structural appropriateness of the non-reduced Petri net results in 0.049, which is an even worse value reflecting the difficulty to understand such a complicated model. The behavioral appropriateness for the desirable scenarios 1 – 5 has been measured to be 0.767, which is a rather good value (for example, the model in Figure 5(e) has a behavioral appropriateness of 0.0 with respect to event log *L2*). But unlike the fitness metric the implemented appropriateness metrics do not indicate an optimal point (such as 1.0 indicates a perfect fitness) and therefore can rather be used as a means to compare alternative process models. This does not apply here but to illustrate the usefulness of a behavioral appropriateness analysis in general, we want to point out that an improved metric should have been able to directly detect the extra behavior covered by Scenario 9, and also to locate the corresponding parts in the model.

7 Related Work

Several attempts have been made to capture the semantics of BPEL [13] in some formal way. Some advocate the use of finite state machines [28], others process algebra [27], abstract state machines [26] or Petri nets [44,32,38,49]. A review of formal semantics of BPEL is given in [44]. This paper uses the translation to Petri nets presented in

[44] which we think is the most detailed one in terms of its coverage of control-flow constructs. We have also developed an approach to translate (Colored) Petri nets into BPEL [5].

This paper builds on earlier work on process mining, i.e., the extraction of knowledge from event logs (e.g., process models [11,12,19] or social networks [8]). For example, the α-algorithm [11] can derive a Petri net from an event log. For an overview of process mining techniques, the reader is referred to [10] and [9]. Process mining can be seen in the broader context of Business Process Intelligence (BPI) and Business Activity Monitoring (BAM). In [31,47] a BPI toolset on top of HP's Process Manager is described. The BPI toolset includes a so-called "BPI Process Mining Engine". In [41] 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) [34]. The tool is commercially available and a customized version of PPM is the Staffware Process Monitor (SPM) [50] which is tailored towards mining Staffware logs.

In this paper we use the conformance checking techniques described in preliminary form in [46] and implemented in the ProM framework [23]. The work of Cook et al. [20,18] is closely related to our work on conformance checking. In [20] the concept of process validation is introduced. It assumes an event stream coming from the model and an event stream coming from real-life observations, both streams are compared. Here the time-complexity is problematic as the state-space of the model needs to be explored. In [18] the results are extended to include time aspects. The notion of conformance has also been discussed in the context of security [6], business alignment [2], and genetic mining [40]. However, in each of the papers mentioned only fitness is considered and appropriateness is mostly ignored.

The need for monitoring web services has been raised by other researchers. For example, several research groups have been experimenting with adding monitor facilities via SOAP monitors in Axis (`http://ws.apache.org/axis/`). [35] introduces an assertion language for expressing business rules and a framework to plan and monitor the

execution of these rules. [15] uses a monitoring approach based on BPEL. Monitors are defined as additional services and linked to the original service composition. Another framework for monitoring the compliance of systems composed of web-services is proposed in [37]. This approach uses event calculus to specify requirements. [36] is an approach based on WS-Agreement defining the Crona framework for the creation and monitoring of agreements. In [30,25], Dustdar et al. discuss the concept of web services mining and envision various levels (web service operations, interactions, and workflows) and approaches. Our approach fits in their framework and shows that web services mining is indeed possible. In [45] a tool named the Web Service Navigator is presented to visualize the execution of web services based on SOAP messages. The authors use Message Sequence Charts (MSCs) and graph-based representations of the system topology. Our work differs from these papers in two ways. First of all, we use a process model to check conformance rather than visualizing and analyzing frequent interaction patterns (i.e. scenarios). Typically, it is easier to specify a process rather than a complete set of scenarios, although scenarios can help in designing and analyzing a process. Moreover, a process specification enables a more intuitive visualization of the problem areas such as deviations from the intended behavior. Second, we consider the problem of correlation in more detail. It is surprising that most prior work skirts this problem.

This paper focuses on conformance by comparing the observed behavior recorded in logs with some predefined model. This could be termed "run-time conformance". However, it is also possible to address the issue of *design-time conformance*, i.e., comparing different process models before enactment. For example, one could compare a specification in abstract BPEL with an implementation using executable BPEL. Similarly, one could check at design-time the compatibility of different services. Here one can use the inheritance notions [3] explored in the context of workflow management and implemented in Woflan [51]. Axel Martens et al. [39,48] have explored questions related to design-time conformance and compatibility using a Petri-net-based approach.

For example, [39] focuses on the problem of static verification of consistency between executable and abstract processes.

8 Conclusion

Service-oriented architectures are composed of relatively autonomous entities (i.e., services). Unlike many classical systems there is not one entity controlling a monolithic system. Therefore, it is essential that each of the services involved in some choreography really behaves as the other services expect it to behave. In this paper we demonstrated the feasibility of choreography conformance checking, i.e., it is indeed possible to monitor services, to detect deviations and measure the degree of conformance.

Although conformance checking of service behavior can be applied to a wide variety of settings, we focused on a particular usage scenario involving (1) abstract BPEL as the specification language of a single service and (2) SOAP messages exchanged between this service and other services. We demonstrated that specifications in terms of abstract BPEL can be mapped onto Petri nets and the SOAP messages exchanged between the various services can be mapped onto our MXML format. This enables us to do conformance checking. Given a set of messages and an abstract BPEL specification we can measure fitness and appropriateness. Moreover, if the observed behavior does not match the specified behavior, the deviations can be shown in both the log and the model. Using a case study utilizing Oracle BPEL as a process engine, we demonstrated that our approach is indeed feasible using current technology. We have implemented three tools to achieve all of this: (1) BPEL2PNML (for the mapping from BPEL to PNML), (2) WofBPEL (for process verification and cleaning up the automatically generated Petri net), and (3) the ProM Conformance Checker.

Although this paper focused on abstract BPEL, it should be noted that other languages could also be supported by replacing BPEL2PNML by a component providing the mapping onto Petri nets for the selected alternative language. In fact, we consider languages such as BPEL and WS-CDL not really suitable for the specification of services. They tend to describe things at a too low level, i.e., a level suitable for execution

but less suitable for describing what the different services need to agree upon. Hence future research will aim at conformance checking in the context of more declarative languages such as DecSerFlow [7] and Let's Dance [52]. Moreover, we would like to apply our approach to more real-life case studies. One of the problems we are facing is that at this point in time only few organizations use BPEL and can provide us with SOAP logs. Clearly, conformance checking can be applied in many domains ranging from auditing (cf. the Sarbanes-Oxley Act) to software testing. Therefore, we plan to consider a wide variety of applications and not limit ourselves to web services. Another topic for further research is the visualization of behavior/conformance, for example, by combining the ideas presented in [45] with our process-oriented approach.

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. 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, Latvia, 2004.
3. W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.
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.
5. W.M.P. van der Aalst, J.B. Jørgensen, and K.B. Lassen. Let's Go All the Way: From Requirements via Colored Workflow Nets to a BPEL Implementation of a New Bank System Paper. 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 22–39. Springer-Verlag, Berlin, 2005.
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 and M. Pesic. DecSerFlow: Towards a Truly Declarative Service Flow Language. In M. Bravetti, M. Nunez, and G. Zavattaro, editors, *International Conference on Web Services and Formal Methods (WS-FM 2006)*, volume 4184 of *Lecture Notes in Computer Science*, pages 1–23. Springer-Verlag, Berlin, 2006.
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. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
13. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.
14. A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, C.K. Liu, S. Thatte, P. Yendluri, and A. Yiu. Web Services Business Process Execution Language Version 2.0. WS-BPEL TC OASIS, 2005.
15. L. Baresi, C. Ghezzi, and S. Guinea. Smart Monitors for Composed Services. In *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*, pages 193–202, New York, NY, USA, 2004. ACM Press.
16. D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/soap, 2000.
17. J. Cardoso, A.P. Sheth, J.A. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *Journal of Web Semantics*, 1(3):281–308, 2004.
18. 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.
19. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
20. 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.
21. F. Curbera, M.J. Duftler, R. Khalaf, W.A. Nagy, N. Mukhi, and S. Weerawarana. Colombo: Lightweight Middleware for Service-Oriented Computing. *IBM Systems Journal*, 44(4):799–820, 2005.
22. 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.
23. B. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau, editors, *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
24. B.F. van Dongen and W.M.P. van der Aalst. A Meta Model for Process Mining Data. In J. Casto and E. Teniente, editors, *Proceedings of the CAiSE'05 Workshops (EMOI-INTEROP Workshop)*, volume 2, pages 309–320. FEUP, Porto, Portugal, 2005.
25. S. Dustdar, R. Gombotz, and K. Baina. Web Services Interaction Mining. Technical Report TUV-1841-2004-16, Information Systems Institute, Vienna University of Technology, Wien, Austria, 2004.
26. D. Fahland and W. Reisig. ASM-based semantics for BPEL: The negative control flow. In D. Beauquier and E. Börger and A. Slissenko, editor, *Proc. 12th International Workshop on Abstract State Machines*, pages 131–151, Paris, France, March 2005.
27. A. Ferrara. Web services: A process algebra approach. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 242–251, New York, NY, USA, 2004. ACM Press.
28. J.A. Fisteus, L.S. Fernández, and C.D. Kloos. Formal verification of BPEL4WS business collaborations. In K. Bauknecht, M. Bichler, and B. Proll, editors, *Proceedings of the 5th International Conference on Electronic Commerce and Web Technologies (EC-Web '04)*, volume 3182 of *Lecture Notes in Computer Science*, pages 79–94, Zaragoza, Spain, August 2004. Springer-Verlag, Berlin.
29. R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.
30. R. Gombotz and S. Dustdar. On Web Services Mining. In M. Castellanos and T. Weijters, editors, *First International Workshop on Business Process Intelligence (BPI'05)*, pages 58–70, Nancy, France, September 2005.
31. 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.
32. S. Hinz, K. Schmidt, and C. Stahl. Transforming BPEL to Petri Nets. In W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, editors, *International Conference on Business Process Management (BPM 2003)*, volume 2678 of *Lecture Notes in Computer Science*, pages 220–235. Springer-Verlag, Berlin, 2003.
33. IBM WebSphere. www.ibm-306.ibm.com/software/websphere.

34. IDS Scheer. ARIS Process Performance Manager (ARIS PPM): Measure, Analyze and Optimize Your Business Process Performance (whitepaper). IDS Scheer, Saarbruecken, Gemany, http://www.ids-scheer.com, 2002.
35. A. Lazovik, M. Aiello, and M. Papazoglou. Associating Assertions with Business Processes and Monitoring their Execution. In *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*, pages 94–104, New York, NY, USA, 2004. ACM Press.
36. H. Ludwig, A. Dan, and R. Kearney. Crona: An Architecture and Library for Creation and Monitoring of WS-agreements. In *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*, pages 65–74, New York, NY, USA, 2004. ACM Press.
37. K. Mahbub and G. Spanoudakis. A Framework for Requirents Monitoring of Service Based Systems. In *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*, pages 84–93, New York, NY, USA, 2004. ACM Press.
38. A. Martens. Analyzing Web Service Based Business Processes. In M. Cerioli, editor, *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering (FASE 2005)*, volume 3442 of *Lecture Notes in Computer Science*, pages 19–33. Springer-Verlag, Berlin, 2005.
39. A. Martens. Consistency between executable and abstract processes. In *Proceedings of International IEEE Conference on e-Technology, e-Commerce, and e-Services (EEE'05)*, pages 60–67. IEEE Computer Society Press, 2005.
40. 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 C. Bussler et al., editor, *BPM 2005 Workshops (Workshop on Business Process Intelligence)*, volume 3812 of *Lecture Notes in Computer Science*, pages 203–215. Springer-Verlag, Berlin, 2006.
41. 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.
42. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
43. Oracle BPEL Process Manager. www.oracle.com/technology/products/ias/bpel.
44. C. Ouyang, W.M.P. van der Aalst, S. Breutel, M. Dumas, , and H.M.W. Verbeek. Formal Semantics and Analysis of Control Flow in WS-BPEL. BPM Center Report BPM-05-15, BPMcenter.org, 2005.
45. W. De Pauw, M. Lei, E. Pring, L. Villard, M. Arnold, and J.F. Morar. Web Services Navigator: Visualizing the Execution of Web Services. *IBM Systems Journal*, 44(4):821–845, 2005.
46. 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, *BPM 2005 Workshops (Workshop on Business Process Intelligence)*, volume 3812 of *Lecture Notes in Computer Science*, pages 163–176. Springer-Verlag, Berlin, 2006.
47. M. Sayal, F. Casati, U. Dayal, and M.C. Shan. Business Process Cockpit. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883. Morgan Kaufmann, 2002.
48. B.H. Schlingloff, A. Martens, and K. Schmidt. Modeling and model checking web services. *Electronic Notes in Theoretical Computer Science: Issue on Logic and Communication in Multi-Agent Systems*, 126:3–26, mar 2005.
49. C. Stahl. Transformation von BPEL4WS in Petrinetze (In German). Master's thesis, Humboldt University, Berlin, Germany, 2004.
50. TIBCO. TIBCO Staffware Process Monitor (SPM). http://www.tibco.com, 2005.
51. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
52. J.M. Zaha, A. Barros, M. Dumas, and A.H.M. ter Hofstede. Lets Dance: A Language for Service Behavior Modeling. QUT ePrints 4468, Faculty of Information Technology, Queensland University of Technology, 2006.