Choreography Conformance Checking: An Approach based on BPEL and Petri Nets

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 Technology Management, 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}@tm.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 Recently, languages such as BPEL and CDL have been proposed to describe the way services can interact from a behavioral perspective. The emergence of these languages heralds an era where richer service descriptions, going beyond WSDL-like interfaces, will be available. However, what can these richer service descriptions serve for? This paper investigates a possible usage of behavioral service descriptions, namely as input for *conformance checking*. Conformance checking is the act of verifying whether one or more parties stick to the agreed-upon behavior by observing the actual behavior, e.g., the exchange of messages between all parties. This paper shows that it is possible to translate BPEL business protocols to Petri nets and to relate SOAP messages to transitions in the Petri net. As a result, Petri net-based conformance checking techniques can be used to quantify *fitness* (whether the observed behavior is possible in the business protocol) and *appropriateness* (whether the observed behavior "overfits" or "underfits" the business protocol). Moreover, non-conformance can be visualized to pinpoint deviations. The approach has been implemented in the context of the ProM framework.

1 Introduction

A Service-Oriented Architecture (SOA) is essentially a collection of services. These services communicate with each other in some meaningful way. To coordinate these services overlapping concepts such as *choreography*, *orchestration*, and *service composition* have been developed. The term "choreography" is typically used to emphasize the fact that the services are autonomous and share the coordination of the overall process. The term "orchestration" typically implies the existence of single coordinating force (the conductor) and the term "service composition" is used to emphasize that larger services are assembled from smaller services. There have been many debates on the differences between these terms. For example, the differences between choreography and orchestration have been

explained using traffic lights (orchestration) and roundabouts (choreography). In our view these distinctions are less meaningful and, therefore, we prefer to focus on concrete languages and choose to use only the term "choreography". The *Business Process Execution Language for Web Services* (BPEL4WS or simply BPEL) has emerged as a standard for specifying and executing processes [21,15]. BPEL can be used to specify so-called *abstract processes*, i.e., a non-executable process specification describing the business protocol as seen from one of the partners involved. The *Web Services Choreography Description Language* (WS-CDL) is another example of a standard allowing for the specification of business protocols, but now seen from a global viewpoint [40].

In this paper we focus on abstract BPEL as a choreography language. However, the concepts are not limited to BPEL. In fact, we advocate the use of more declarative choreography languages as discussed in [4].

We assume that a language like BPEL is used to describe the desired behavior. Although a BPEL specification could be used as an execution language for a given (composite) service, the service and the services it interacts with do not need to be implemented using BPEL. They could use other languages and the internal behavior of external services may be invisible. Note that partners have no control over each other's services. Moreover, partners will typically not expose the internal structure and state of their services. This triggers the question of *conformance*: "Do all parties involved operate as described?". We will use the term *choreography conformance checking* to refer to this question. To address the question we assume the existence of both a *process model* (describing the desired choreography) and an *event log* (describing the actual observed behavior). As already indicated in [4,27,31,35] it seems possible to extract event logs in a SOA by monitoring the exchange of messages or transaction logs.

Choreography conformance checking benefits from the coexistence of event logs and process models and may be viewed from two angles. 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 "incorrect" event sequences not possible according to the model. This may indicate violations of choreography all parties previously agreed upon. 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.

Figure 1 describes the approach proposed in this paper. Based on a process model described in terms of abstract BPEL, we automatically create a Petri net [22,23] description of the intended choreography. We use our translation described in [51] and implemented in our tool BPEL2PNML.[1] We also describe an approach to monitor SOAP messages and transform these into events logs. The event logs are stored in the so-called MXML format [25]. By comparing an event log and a Petri net we enable conformance checking. To actually measure

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

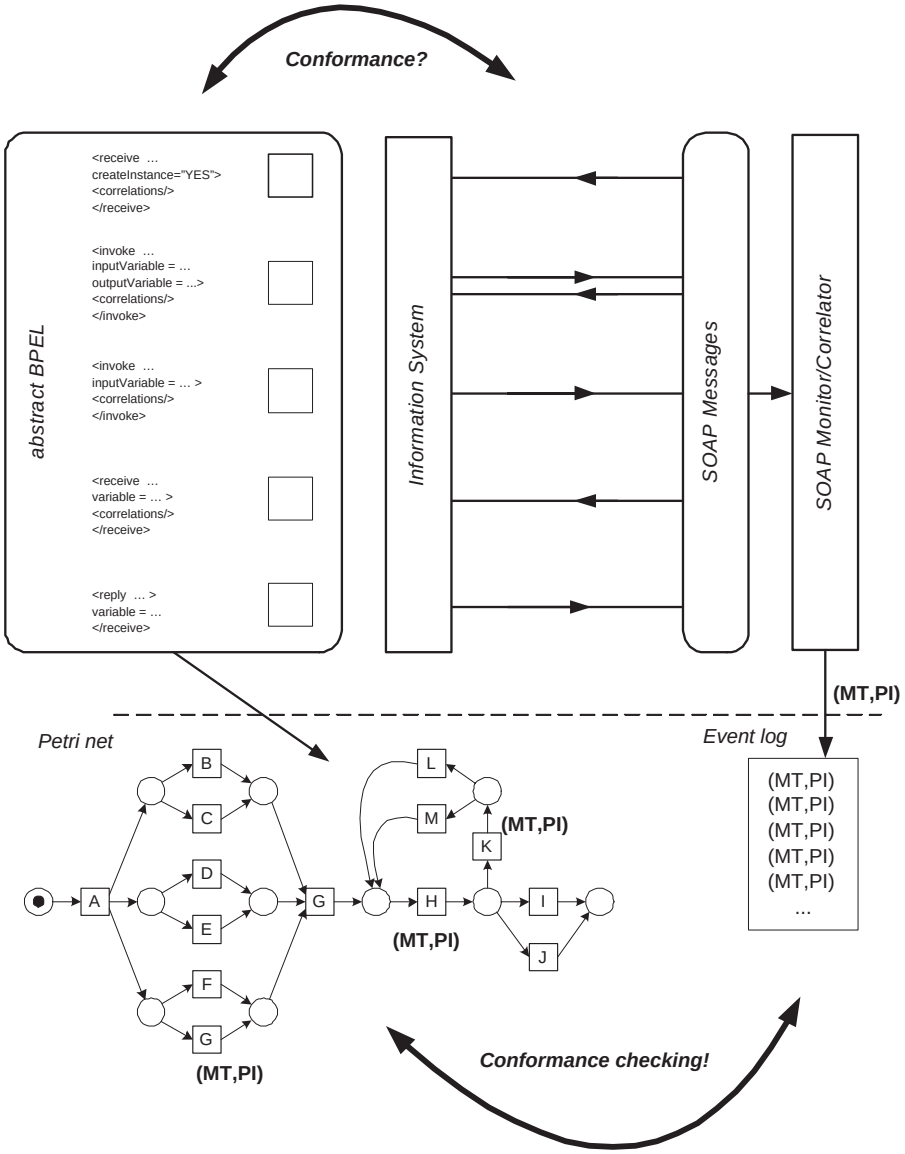


Figure 1. Overview of the approach implemented.

conformance, we have developed a tool called the *Conformance Checker*. This tool has been developed in the context of the *ProM framework*[2]. Although ProM offers a wide range of tools related to process mining (e.g., LTL checking, process discovery, extracting other models from event logs, verification, etc.) [10,12], we focus on the Conformance Checker described in [54].

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 describing the choreography should be able to "parse" every event sequence observed by monitoring e.g. SOAP messages. In [54] 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. 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. Therefore, we use 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". The ProM Conformance Checker supports both the notion of fitness and the notion of appropriateness.

The remainder of this paper is organized as follows. First, we elaborate on the concept of choreography conformance checking. Section 3 presents the notion of conformance checking and introduces the functionality of the ProM Conformance Checker. Then we discuss the mapping of BPEL [51] onto WF-nets [1], a subclass of Petri nets. A running example is used to illustrate the mapping. Moreover, two tools as discussed: a tool to automatically translate BPEL onto Petri nets (BPEL2PNML) and a tool to analyze and reduce the resulting net (WofBPEL). Section 5 discusses the various ways in which it is possible to extract event logs from messages (e.g., using SOAP headers). Section 6 describes a case study demonstrating the feasibility of our approach and the tools we have developed. For this case study we use Oracle BPEL as a process engine. Related work is discussed in Section 7. Finally, Section 8 concludes the paper.

2 Conformance Checking of Choreographies

As indicated in the introduction, this paper focuses on choreography conformance checking using "SOAP-messages based event logs" and "abstract BPEL based Petri nets". However, before introducing the notion of conformance checking and the mapping from BPEL to Petri nets, we consider choreography conformance checking in a broader perspective. To do this we discuss four possible settings for choreography conformance checking as shown in Figure 2. Each of the four settings describes a SOA where four services are interacting by exchanging messages. Each service contains multiple activities involved in some interaction (e.g., an `invoke`, `receive` or `reply`). Only the activities/messages indicated in bold are visible.

Figure 2(a) shows the setting where there is some global observer that can monitor all messages. This could be realized by implementing some message

[2] Both documentation and software can be downloaded from `www.processmining.org`.

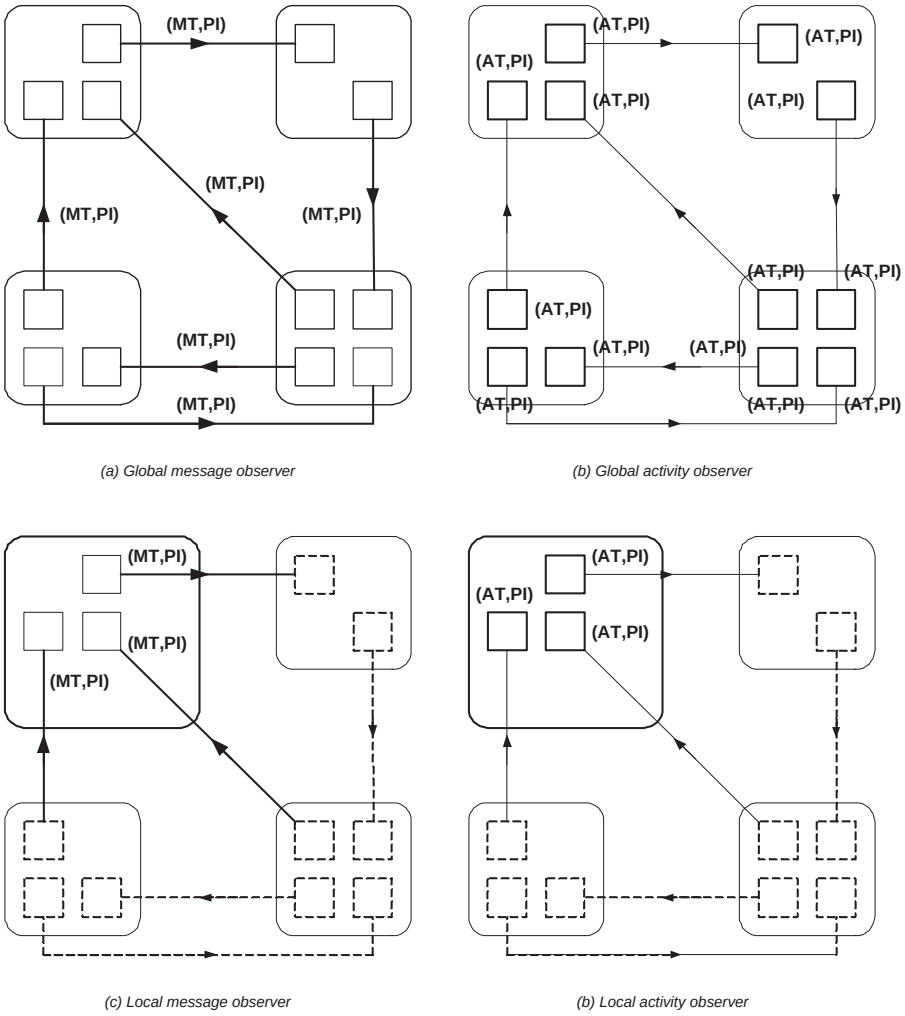


Figure 2. Four possible settings for choreography conformance checking: (a) relevant messages exchanged between all services involved in a choreography are visible, (b) relevant activities executed inside all services involved in a choreography are visible, (c) relevant messages exchanged with a single service are visible, and (d) relevant activities executed within a single service are visible. (MT = Message Type, AT = Activity Type, and PI = Process Instance).

broker that connects all services. Similarly (but less realistic), it could be possible to globally monitor the execution of events, cf. Figure 2(b). Note that in many SOA platforms (e.g., IBM's Websphere) it is possible to monitor activities. Figure 2(c) describes the setting where all messages exchanged with a specific service are recorded. Note that the observer does not need to view the process globally, a local monitoring facility suffices. It is often also possible to monitor activities from such a local perspective, cf. Figure 2(d). In this paper, *we will focus on the setting illustrated by Figure 2(c).*

For conformance checking,[3] it is crucial that each event recorded in the log can be linked to some process instance (i.e., case) and some model element (e.g., an activity in BPEL terms or a transition in Petri-net terms). This is reflected in Figure 2 by *(MT,PI)* (*MT* is the message type that can be linked to some activity and *PI* refers to a specific process instance) and *(AT,PI)* (*AT* refers to some activity). This may seem trivial but in reality it is not. At any point in time there may be an arbitrary number of process instances all exchanging messages 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.

As indicated above, we focus on the setting illustrated by Figure 2(c) and assume that we are able to map messages onto events of the form *(MT,PI)*. Moreover, we assume that there is an abstract process specified in terms of BPEL [21,15]. Note that a BPEL specification may be *descriptive* or *prescriptive*. Abstract BPEL processes are of a descriptive nature since they do not enforce the service to operate in a certain way. Executable BPEL processes are of a prescriptive nature since they are used by some process engine to enforce a particular way of operating. The focus of this paper is on conformance checking and therefore we only assume the existence of an abstract BPEL process without making any assumptions on the implementation of the process logic in the service. Figure 3 illustrates the setting we are focusing on. There is some abstract BPEL process and observed messages can be linked to BPEL activities such as `receive`, `invoke`, 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 also be discrepancies between the left and the right hand parts of Figure 3. 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 a BPEL setting, it is important to note that also other (more declarative) languages could be used. The Web Services Choreography Description Language (WS-CDL) would be an obvious candidate

[3] In fact, for process mining [10,12] in general it is important that each event points to some process instance and a model element.

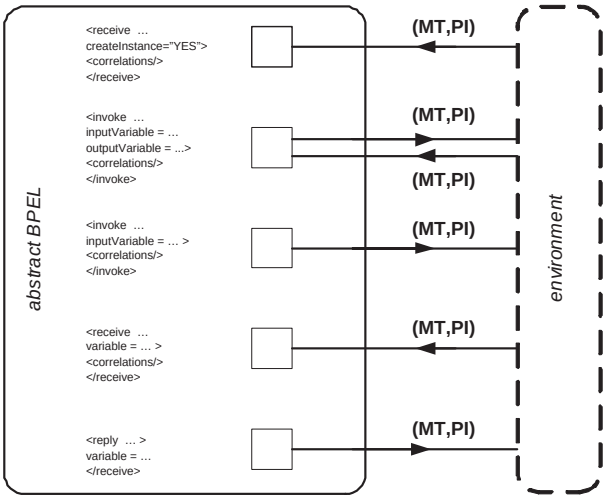


Figure 3. The situation illustrated by Figure 2(c) put into the context of BPEL with its basic activities `receive` (initial or not), `invoke` (synchronous or asynchronous), and `reply`.

[40]. The process part of WS-CDL can be seen as a subset of BPEL and therefore it would be easy to extend this work to WS-CDL. However, as indicated in [4], we would prefer a more declarative language. Since such a more declarative standard is missing at this point in time, we focus on abstract BPEL in this paper.

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 [53,54] this question has been explored using Petri nets to represent process models [22,23], 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 figures 2 and 3), (ii) refer to a case (i.e., a process instance, cf. labelled PI in figures 2 and 3), and (iii) be totally ordered.

We have identified two dimensions of conformance, *fitness* and *appropriateness* [53,54]. 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).

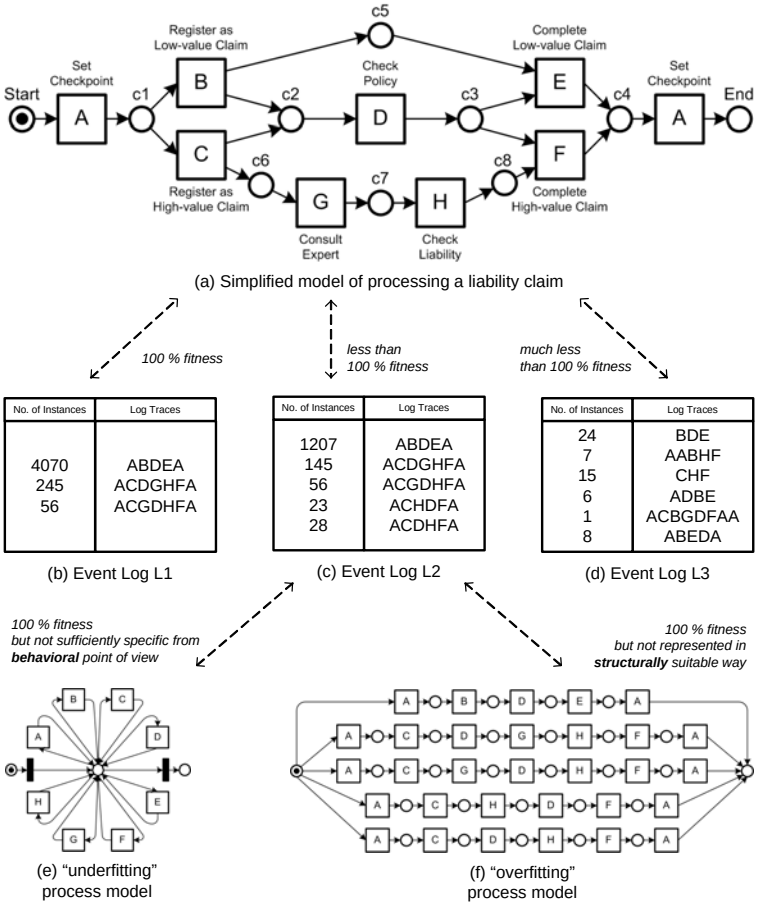


Figure 4. Two dimensions of conformance: fitness and appropriateness.

To illustrate both dimensions of conformance we use the example process shown in Figure 4(a).[4] At first there are two activities bearing the same label "Set Checkpoint". 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 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

[4] For simplicity we use a simple example without any reference to SOA. Later we will show examples involving multiple services.

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 4(b)-(d) show three example logs for the process described in Figure 4(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 [9], the resources performing an activity would distinguish those instances from each other.

Event log *L1* completely *fits* the model in Figure 4(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 even contain one trace corresponding to the specified behavior.

Now consider the two process models shown in Figure 4(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 does not allow for more sequences than those having been observed but 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 4. 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 [53,54]. 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 logical 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 had been left in the model, which indicates the process not having *properly completed*. Only if there were neither tokens left nor missing the fitness measure evaluates to 1, which indicates 100 % fitness.

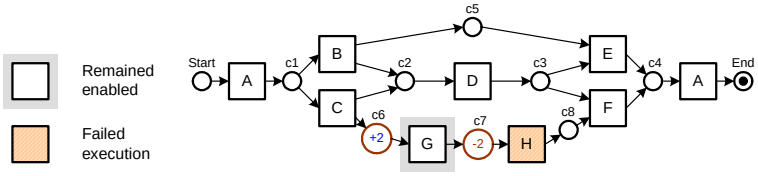


Figure 5. Example process model after replay of event log *L2*.

Figure 5 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 [53,54]. Note that the checker supports *duplicate activities*, e.g., in Figure 4(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 is 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 recently developed by the authors of this paper: BPEL2PNML and WofBPEL. BPEL2PNML translates BPEL process definitions into Petri nets represented in the Petri Nets Markup Language (PNML). WofBPEL, built using Woflan [59,61], 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 6 provides an overview of a "Supplier Service" using a visual notation that directly reflects 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.

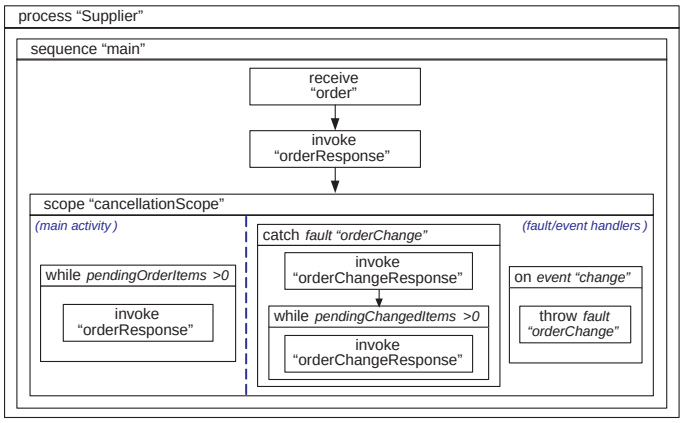


Figure 6. An abstract view of the Supplier process.

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.

Appendix A contains the definitions of the Supplier service both as an abstract and as an executable BPEL process. 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 language (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 2. This enables conformance checking based on the activity-oriented logs as illustrated by figures 2(b) and 2(d). Otherwise, conformance checking in the style of figures 2(a) and 2(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 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) 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 [51].

Activities We start with the mapping of a basic activity (X) shown in Figure 7, which also illustrates our mapping approach for structured activities. The net is divided into two parts: one (drawn in solid lines) models the normal processing of X, the other (drawn using dashed lines) models the skipping of X.

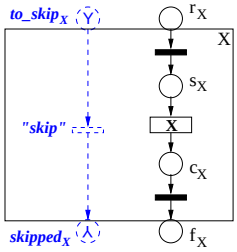


Figure 7. A Basic activity.

In the normal processing part, the four places are used to capture four possible states for the execution of activity X: r_X for "ready" state, s_X for "started" state, c_X for "completed" state, and 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 internal actions for checking preconditions or evaluating post-conditions for activities. The skip path is mainly used to facilitate the mapping of control links. Note that the to_skip and 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 7, hiding the subnet enclosed in the box labeled X yields an abstract graphic representation of the mapping for activities. This will be used in the rest of the paper.

Figure 8 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 8 and subsequent figures, the skip path of the mapping is not shown if it is not used. A detailed description of the mapping [51] 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.

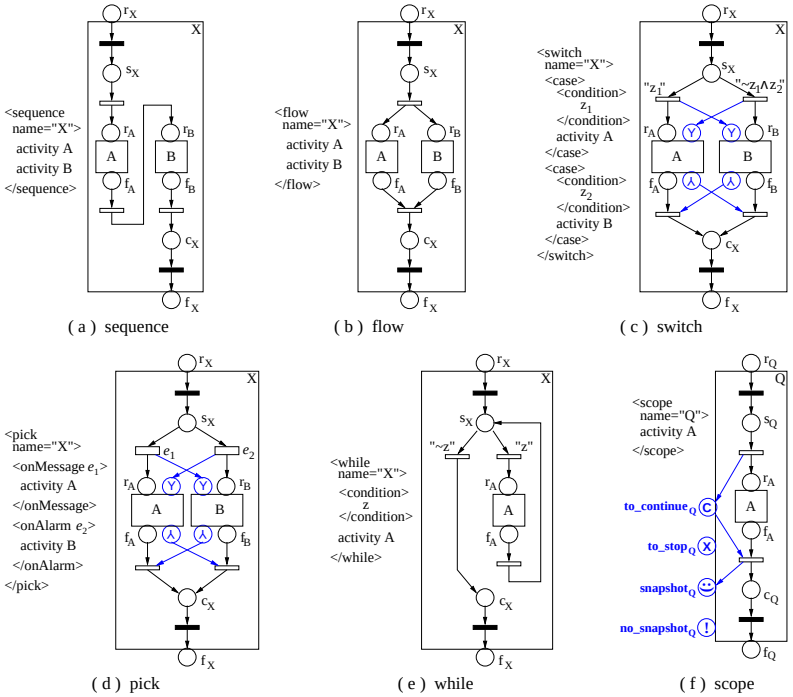


Figure 8. Structured activities.

A *while* activity supports structured loops. In Figure 8(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 ($\sim$z).

A *scope* provides event and exception handling. It has a main activity that defines its normal behavior. To facilitate the mapping of exception handling, we define four flags for a scope, as shown in Figure 8(f). These 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 [15] which refers to the preserved state of a successfully completed uncompensated scope; and no_snapshot, indicating the absence of a scope snapshot.

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 9 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 e_{normal} is *enabled* and may occur upon an environment or a system trigger. When e_{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 e_{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 e_{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.

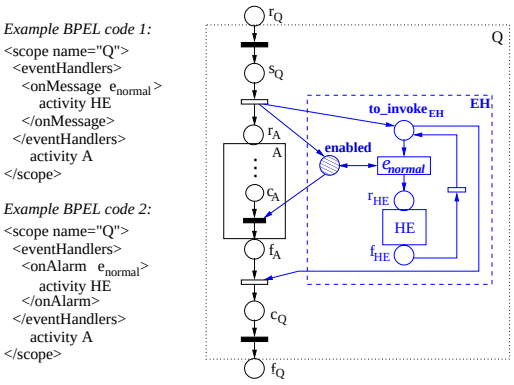


Figure 9. An event handler.

Fault Handlers If a fault occurs during the normal process of 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. A scope in which a fault has occurred is considered to have ended abnormally and thus cannot be compensated, no matter whether or not the fault can be handled successfully. Figure 10 depicts the mapping of a scope (Q) with a fault handler (FH). The subnet enclosed in the box labeled FH specifies the mapping of FH. It has a

similar structure as the mapping of an event handler. Nevertheless, there are some subtle differences as explained next.

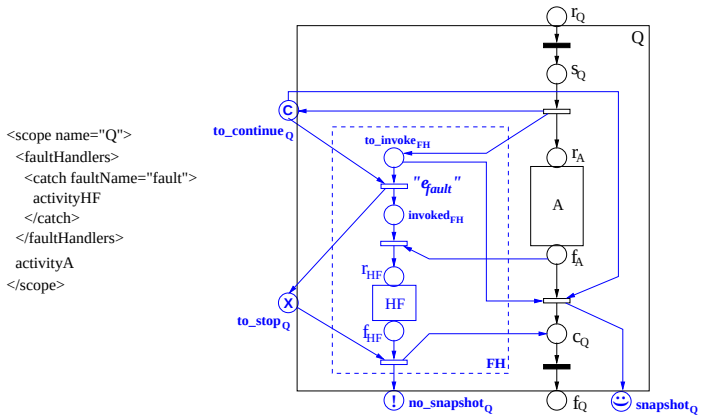


Figure 10. A fault handler.

Firstly, as compared to the normal events defined within event handlers, faults that may arise during a process execution can be considered as *fault events*. Transition "$\mathsf{e_{fault}}$" represents such fault event, and upon its occurrence, the status of scope Q changes from to_continue to to_stop. All activities that are currently active in Q need to stop, and any other fault events that may occur are disabled.

Secondly, the fault handler FH, once invoked, cannot start its activity HF ("handling fault") until the main activity (A) of scope Q has terminated. Place $\mathsf{invoked_{FH}}$ is used to capture an intermediate state after the occurrence of $\mathsf{e_{fault}}$ but before the execution of HF. For the mapping of activity termination (which is not shown in Figure 10), we adopt an approach of conducting a dry run of the activity without performing its concrete actions (i.e., the core action of each basic activity) nor allowing it to process any normal event. As a result, an activity being terminated will end up in the "finished" state. For example, in Figure 10, if activity A is required to terminate, place $\mathsf{f_A}$ will get marked upon its termination.

Finally, if the fault has been handled successfully, evaluation of any post-condition for scope Q will be performed as normal. Accordingly, in Figure 10, place $\mathsf{c_Q}$ will get marked. However, the status of Q will change from to_stop to no_snapshot to indicate that a fault has occurred during its performance.

Example: Mapping of the Supplier Process Figure 11 depicts the mapping of the Supplier process shown in Figure 6. For illustration purposes, some net details (e.g., those associated to the process scope and the skip paths) are omitted. The complete mapping of the Supplier process, as obtained using BPEL2PNML, is shown in Figure 12. Below, we use Figure 11 to illustrate the mapping.

In Figure 11, We instantiate the general mapping of a fault handler shown in Figure 10 for a fault generated by the throw activity (e.g., *throwFault*). In this case, the action of throwing a fault itself resembles the corresponding fault event. Also, the mapping in Figure 11 ensures that, when the *cancellationScope* is faulty, 1) any active instance of event handler will terminate before the fault handling starts; 2) event *change* is disabled; and 3) for activity *while1*, its sub-activity *orderResponse* is dry-runned once.

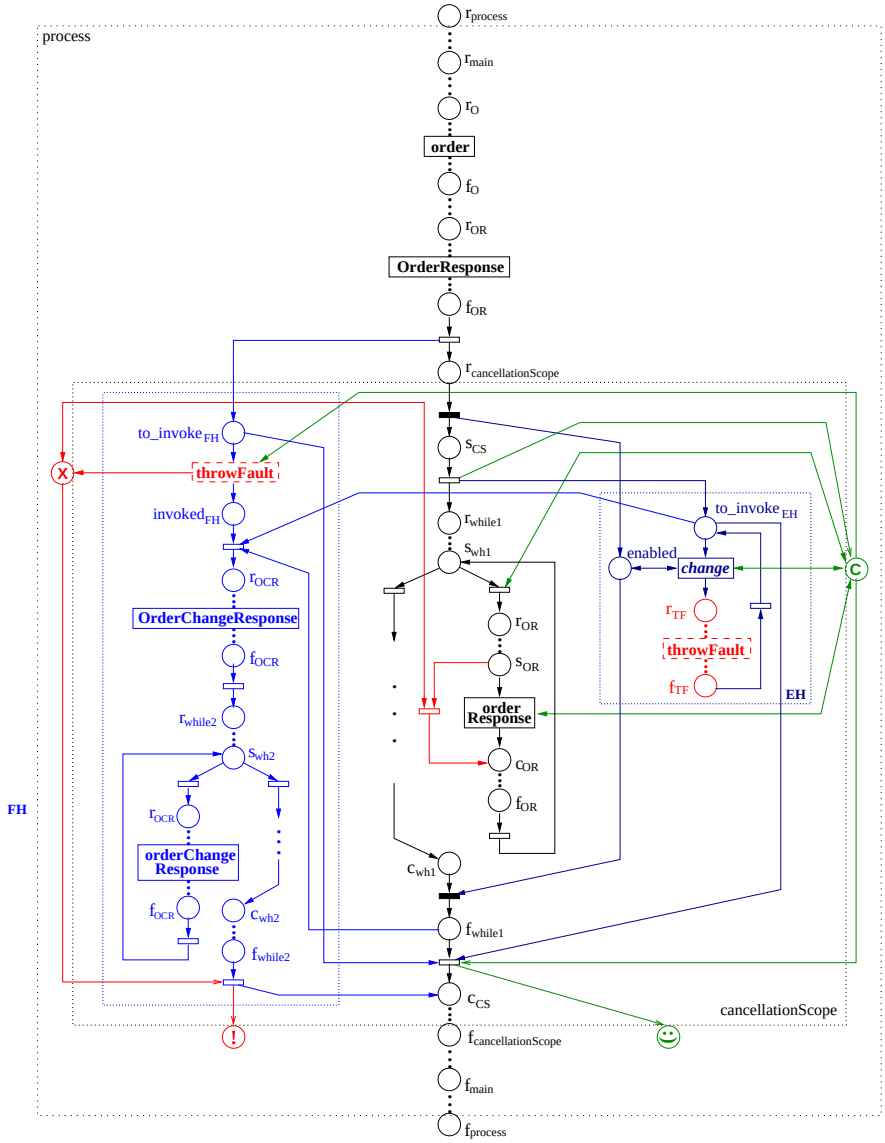
4.3 From Petri nets to WF-nets

The ProM Conformance Checker takes a WF-net [1] and an MXML log [25] 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].

The Petri net shown in Figure 12 is not a WF-net. It contains unconnected parts and additional source and sink places resulting from idle skip fragments (e.g., the two places connected via a τ-transition located on the top right of Figure 12). In order to facilitate the mapping of control links 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 get skipped. A straightforward counter example is the root activity (i.e., the process scope). Thus, by removing these idle skip fragments, the above Petri net 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 as output from BPEL2PNML. Since it uses Woflan [59,61] and Woflan can only handle WF-nets, WofBPEL has been provided with the functionality to remove the idle skip fragments. In addition, WofBPEL has also been provided with behavior preserving reduction rules based the ones as given by Murata [50], and therefore can be used to reduce the size of a net as much as possible (by removing transient states). 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 not be removed.

Figure 13 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 rules are self-explanatory. However, when applying the rules one should clearly differentiate between silent and non-silent transitions. For example, in the fourth rule



[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 6.

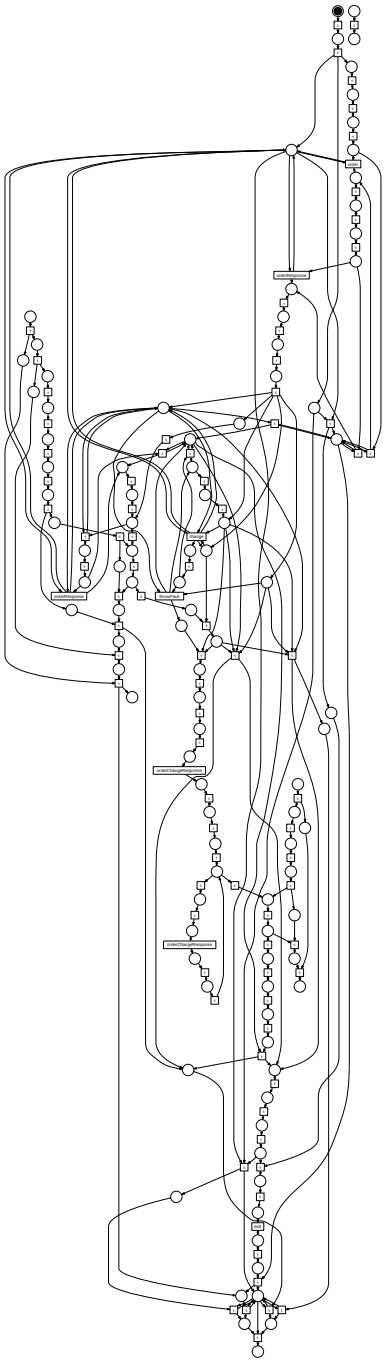


Figure 12. Mapping of the Supplier process as output from BPEL2PNML.

at least one of the transitions should be silent, otherwise the rule should not be applied (as indicated). Note that in this rule the execution of y is inevitable once x has been executed. Therefore, it is only possible to postpone its occurrence. The reduction rules shown in Figure 13 do not preserve the moment of choice and therefore assume trace semantics rather than branching/weak bisimulation [34].

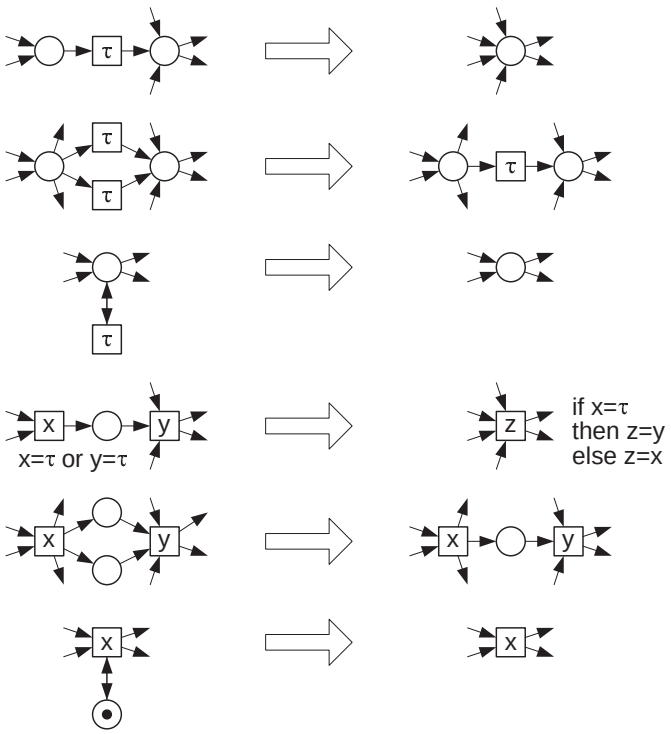


Figure 13. Behavior preserving reduction rules used in WofBPEL.

Based on the above, Figure 14 depicts the WF-net resulting from the Petri net shown in Figure 12. It has been automatically generated from the BPEL process shown in the Appendix using BPEL2PNML and WofBPEL.

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

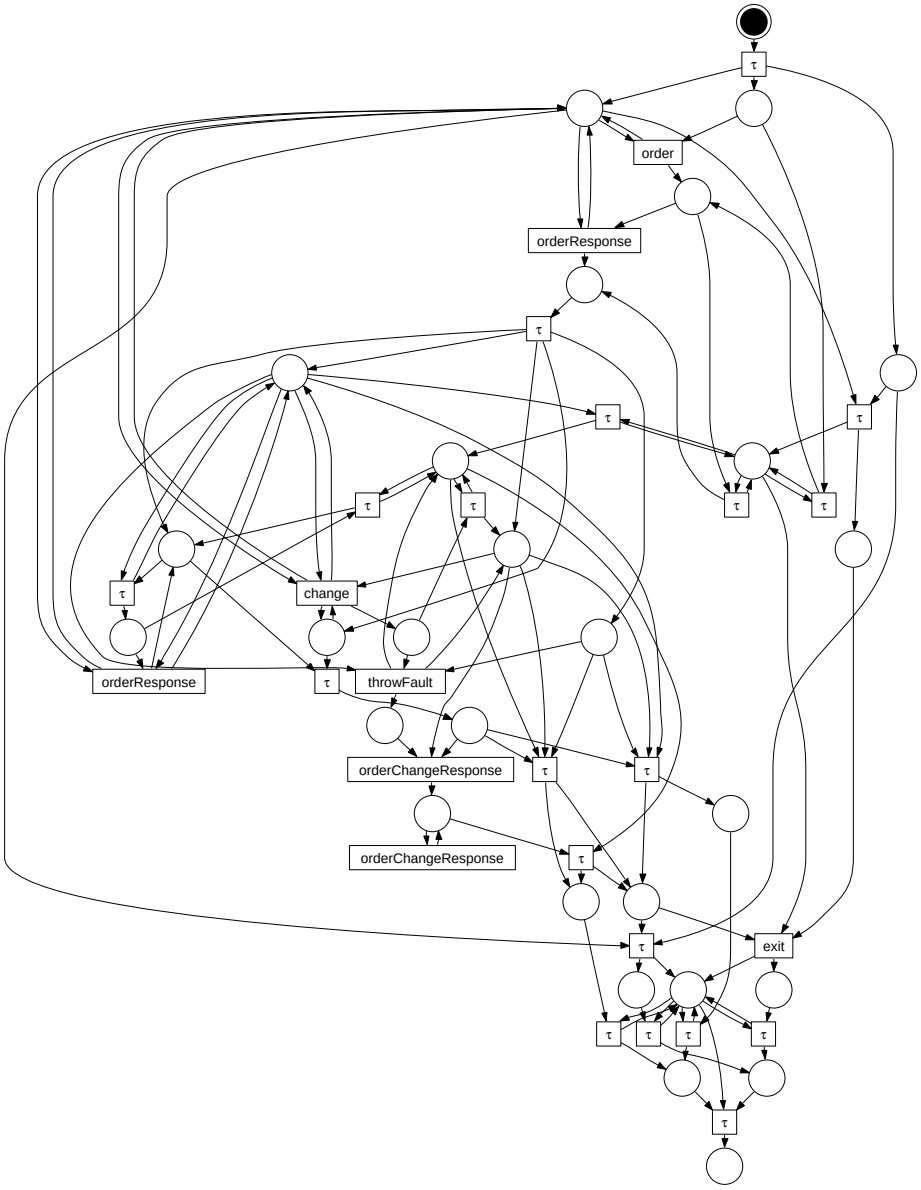


Figure 14. The WF-net for the Petri net shown in Figure 12.

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 4(b)-(d).[5] 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 2, these labels must allow one to determine the direction of messages (indicated by arrows in Figure 2) and its message type (designated as MT in Figure 2). 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.

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 (case). If the service is implemented as an executable BPEL process, this grouping of messages is trivial. The executable 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 given 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.[6] However, we can use an alternative 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

[5] 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.

[6] Note that more monitoring approaches in the web-services context struggle with the same problem. Therefore, dedicated platforms have been developed to deal with this problem. For example the Web Services Navigator [52] uses IBM's Data Collector logging the content and context of SOAP messages. The Data Collector inserts a proprietary SOAP header element into messages that enables correlation.

service standards and platforms 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 implementation and execution environments.
- 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 2(a) where the message logs and an abstract BPEL process are available.

5.2 Abstracting messages as labels

Once the message log has been grouped into log traces corresponding to service instances, we need to associate each message in a log trace with a transition label used in the Petri 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 2. 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 then get translated to silent Petri nets 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 4(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 in 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 communication protocol, but in any case it provides a means to identify messages that pertain to that operation. In the case of SOAP over HTTP, the binding information associates the WSDL operation to a *SOAP-action* identifier. This makes it possible to associate a message with a WSDL operation by inspecting the SOAP-action field in the HTTP header of the message. In the case of a communication protocol based on plain XML over HTTP, the binding information includes a mapping from each WSDL operation to a relative URL to be found in the HTTP headers of every message pertaining to that operation. This makes it possible to associate a message to an operation by analyzing the "request URI" in the message's HTTP header.

Thus, 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 2(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 15 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 16 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. For sake of generality, we ran both processes on two different (Oracle BPEL 10.1.2) servers.

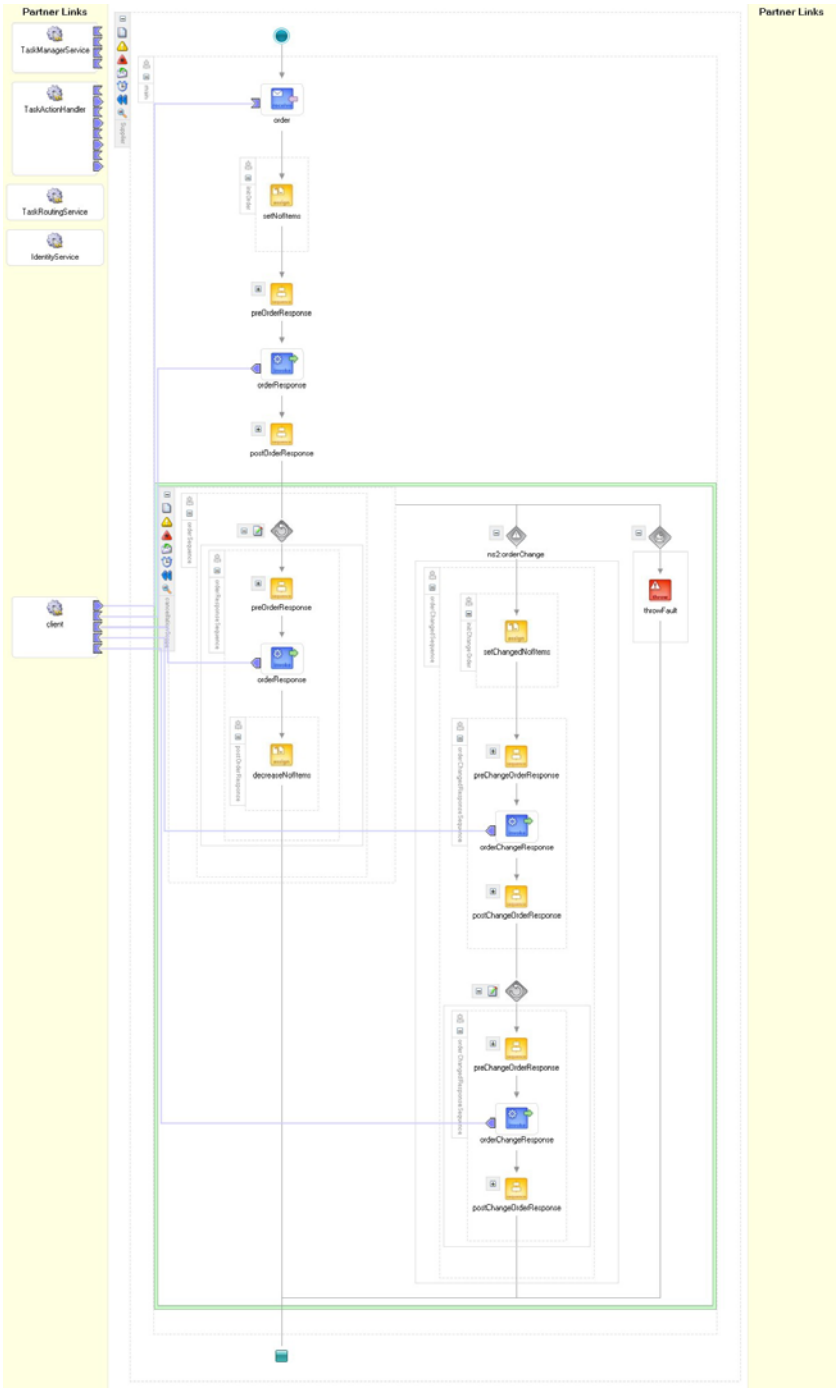


Figure 15. The Supplier process using JDeveloper.

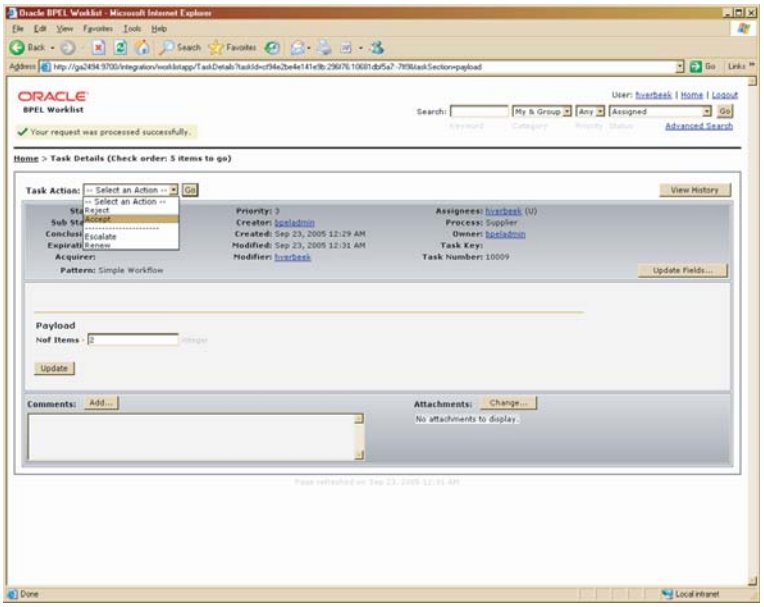


Figure 16. A work item in Oracle BPEL Worklist.

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[7] 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 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 Appendix B. The Customer process was run on the server named ga2550, whereas the Supplier process was run on the server named ga2494.

6.3 Message correlation

From the SOAP message logs (see Appendix B), it is straightforward to generate a log as shown in Figure 17: 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.

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

Both the WF-net corresponding to the abstract Supplier process and the log from Figure 17 can be imported by the ProM framework to check their conformance.[8]

6.4 Conformance checking

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)

Having demonstrated that it is feasible to obtain such an event log 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 17). 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 correlated to a previous *order*, and in Scenario 9 it still sends another *orderResponse* although a

[8] 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`.

```
<?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 17. An example SOAP-based log.

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 14. Having imported it into the ProM framework, the Conformance Checker [53,54] 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 diagnostic visualization of missing and remaining tokens may help to locate the problem.

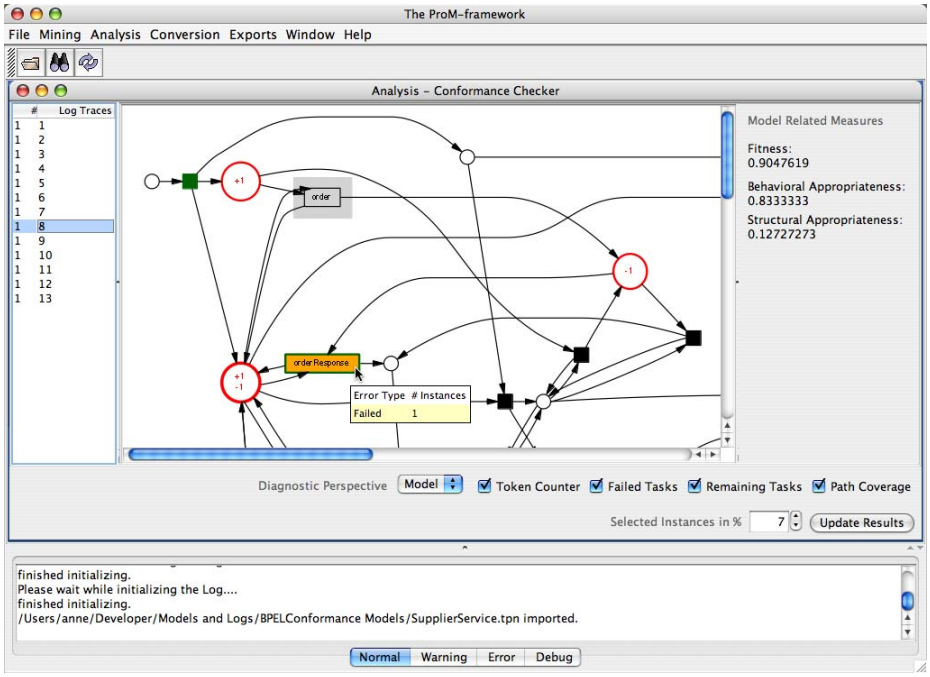


Figure 18. The Conformance Checker analyzing the scenarios from Table 1.

Consider for example Figure 18, 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 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 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 [53,54]). As for the structural appropriateness, the reduced Petri net depicted in Figure 14 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 4(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

Since the early nineties, workflow technology has matured [33] and several textbooks have been published, e.g., [5,26]. Petri nets have been used for the modeling of workflows [5,17,26] but also the orchestration of web services [47].

Several attempts have been made to capture the behavior of BPEL [14] in some formal way. Some advocate the use of finite state machines [30], others process algebras [29], and yet others abstract state machines [28] or Petri nets [51,44,57,60]. (See [51] for a more detailed literature review.) For a detailed analysis of BPEL based on the workflow patterns [6] we refer to [62]. This paper uses the the translation presented in [51]. Note that we have also developed an approach to translate (Colored) Petri nets into BPEL [7].

Clearly, this paper builds on earlier work on process mining, i.e., the extraction of knowledge from event logs (e.g., process models [12,13,19,31,32,38] or social networks [9]). For example, the well-known α algorithm [12] can derive a Petri net from an event log. It is impossible to give a complete overview of process mining here. Therefore, we refer to a special issue of Computers in Industry on process mining [11] and a survey paper [10]. Process mining can be seen in the broader context of Business (Process) Intelligence (BPI) and Business Activity Monitoring (BAM). In [36,37,55] a BPI toolset on top of HP's Process Manager is described. The BPI toolset includes a so-called "BPI Process Mining Engine". In [49] 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) [39]. The latter tool is commercially available and a customized version of PPM is the Staffware Process Monitor (SPM) [58] which is tailored towards mining Staffware logs.

In this paper we use the conformance checking techniques described in [53,54] and implemented in our ProM framework [24]. 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 [8], business alignment [2], and genetic mining [48]. However, in each of the papers mentioned only fitness is considered and appropriateness is mostly ignored. An exception is the preliminary work reported in [53,54]. Therefore, this style of conformance checking is used in this paper.

The need for monitoring web services has been raised by other researchers. For example, several research groups have been experimenting with adding mon-

itor facilities via SOAP monitors in Axis `http://ws.apache.org/axis/`. [41] introduces an assertion language for expressing business rules and a framework to plan and monitor the execution of these rules. [16] 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 [43]. This approach uses event calculus to specify requirements. [42] is an approach based on WS-Agreement defining the Crona framework for the creation and monitoring of agreements. In [35,27], 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 [52] 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 diagrams 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 a set of rules. Typically, it is easier to specify a process rather than a complete set of constraints. Moreover, it enables a more intuitive visualization of the problem areas. Second, we consider the problem of correlation (i.e., linking messages to a single process instance) in more detail. It is surprising that most papers simply ignore 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 [61]. Axel Martens et al. [44,45,46,56] have explored questions related to design-time conformance and compatibility using a Petri-net-based approach. For example, [45] focuses on the problem of consistency between executable and abstract processes and [46] presents an approach where for a given composite service the required other services are generated.

8 Conclusion

In this paper we explored the feasibility of choreography conformance checking. We started with an overview describing the various ways one could extract run-time data in the context of services. For a particular language, i.e., abstract BPEL, we showed that it is possible to do conformance checking. 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 appropriate-

ness. 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. This paper focused on abstract BPEL. However, note 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 our conformance checking experiments we experienced that the use BPEL as a specification language (abstract BPEL) is problematic. BPEL is essentially an imperative programming language in which the basic communication primitives such as sending and receiving a message are mixed with a wide variety of routing constructs. The same process can be represented in different ways[9], but in most cases the user is forced to over-specify the desired behavior. For example, it is not possible to specify that two activities exclude each other (this is different from a choice made at some point). Similar problems apply to WS-CDL, which fundamentally only differs from BPEL in that it views interactions from a global perspective. As a result, these languages are unsuitable for describing processes to be used as a starting point for conformance checking. Indeed, conformance checking is more useful when performed against specifications of what a service must, can and cannot do, rather than against specific procedures for delivering services which is what WS-CDL and BPEL capture. A "true" choreography language should allow for the specification of the "what" without having to state the "how". Also, in such language, being able to express what cannot happen is as important as being able to express what can happen, and both BPEL and WS-CDL fail to capture "forbidden" scenarios: they focus exclusively on possible scenarios. This is similar to the difference between a program and its specification. One can specify what an ordered sequence is by formulating a set of properties, without specifying an algorithm to sort a sequence.

Future work will aim at the development of a better language for specifying choreographies. 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 do a wide variety of applications and not limit ourselves to web services. Another topic for further research is the visualization of behavior/conformance, e.g., we would like to combine the ideas presented in [52] with our more process-oriented approach.

[9] Recall that BPEL merges two styles of modeling, i.e., the block-structured style of Microsoft's Biztalk and the graph-oriented style of IBM's MQSeries Workflow.

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, A.H.M. ter Hofstede, N. Russell, H.M.W. Verbeek, and P. Wohed. Life After BPEL? In M. Bravetti, L. Kloul, and G. Zavattaro, editors, *WS-FM 2005*, volume 3670 of *Lecture Notes in Computer Science*, pages 35–50. Springer-Verlag, Berlin, 2005.
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, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.

14. 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.
15. 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.
16. 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.
17. P. Chrzastowski-Wachtel. A Top-down Petri Net Based Approach for Dynamic Workflow Modeling. 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 336–353. Springer-Verlag, Berlin, 2003.
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, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, 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.
22. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
23. 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.
24. 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.
25. 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.
26. M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Wiley & Sons, 2005.
27. 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.

28. 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.
29. 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.
30. 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.
31. W. Gaaloul, S. Bhiri, and C. Godart. Discovering Workflow Transactional Behavior from Event-Based Log. In R. Meersman, Z. Tari, W.M.P. van der Aalst, C. Bussler, and A. Gal et al., editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2004*, volume 3290 of *Lecture Notes in Computer Science*, pages 3–18, 2004.
32. W. Gaaloul and C. Godart. Mining Workflow Recovery from Event Based Logs. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Business Process Management (BPM 2005)*, volume 3649, pages 169–185. Springer-Verlag, Berlin, 2005.
33. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.
34. R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.
35. 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.
36. 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.
37. 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.
38. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
39. 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.
40. N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web Services Choreography Description Language Version 1.0(W3C Working Draft 17 December 2004). http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/, 2004.
41. 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.

42. 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.
43. 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.
44. 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.
45. 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.
46. P. Massuthe, W. Reisig, and K. Schmidt. An Operating Guideline Approach to the SOA. In *Proceedings of the 2nd South-East European Workshop on Formal Methods 2005 (SEEFM05)*, Ohrid, Republic of Macedonia, 2005.
47. M. Mecella, F. Parisi-Presicce, and B. Pernici. Modeling E-service Orchestration through Petri Nets. In *Proceedings of the Third International Workshop on Technologies for E-Services*, volume 2644 of *Lecture Notes in Computer Science*, pages 38–47. Springer-Verlag, Berlin, 2002.
48. A.K.A. de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Using Genetic Algorithms to Mine Process Models: Representation, Operators and Results. BETA Working Paper Series, WP 124, Eindhoven University of Technology, Eindhoven, 2004.
49. 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.
50. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
51. 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-05-15, BPMcenter.org, 2005.
52. 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.
53. A. Rozinat. Conformance Testing: Measuring the Alignment Between Event Logs and Process Models. Master's thesis, Hasso-Plattner-Institute for Software Engineering at University and Eindhoven University of Technology, Eindhoven, The Netherlands, 2005.
54. A. Rozinat and W.M.P. van der Aalst. Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In M. Castellanos and T. Weijters, editors, *First International Workshop on Business Process Intelligence (BPI'05)*, pages 1–12, Nancy, France, September 2005.
55. 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.

56. 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.
57. C. Stahl. Transformation von BPEL4WS in Petrinetze (In German). Master's thesis, Humboldt University, Berlin, Germany, 2004.
58. TIBCO. TIBCO Staffware Process Monitor (SPM). http://www.tibco.com, 2005.
59. H.M.W. Verbeek and W.M.P. van der Aalst. Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 475–484. Springer-Verlag, Berlin, 2000.
60. H.M.W. Verbeek and W.M.P. van der Aalst. Analyzing BPEL Processes using Petri Nets. In D. Marinescu, editor, *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 59–78. Florida International University, Miami, Florida, USA, 2005.
61. 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.
62. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In I.Y. Song, S.W. Liddle, T.W. Ling, and P. Scheuermann, editors, *22nd International Conference on Conceptual Modeling (ER 2003)*, volume 2813 of *Lecture Notes in Computer Science*, pages 200–215. Springer-Verlag, Berlin, 2003.

A Details of Supplier BPEL process

This appendix contains the WSDL file, the abstract BPEL file, and the executable BPEL file for the Supplier process. To improve readability, we have inserted line breaks.

A.1 WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Supplier"
    targetNamespace="http://services.qut.com/Supplier"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:client="http://services.qut.com/Supplier"
    xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">

    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    TYPE DEFINITION - List of services participating in this BPEL process
    The default output of the BPEL designer uses strings as input and
    output to the BPEL Process. But you can define or import any XML
    Schema type and us them as part of the message types.
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <types>
        <schema attributeFormDefault="qualified"
            elementFormDefault="qualified"
            targetNamespace="http://services.qut.com/Supplier"
```

```
        xmlns="http://www.w3.org/2001/XMLSchema">
        <element name="SupplierProcessRequest">
            <complexType>
                <sequence>
                    <element name="nofItems" type="integer"/>
                </sequence>
            </complexType>
        </element>
        <element name="SupplierProcessResponse">
            <complexType>
                <sequence>
                    <element name="nofItems" type="integer"/>
                </sequence>
            </complexType>
        </element>
    </schema>
</types>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MESSAGE TYPE DEFINITION - Definition of the message types used as
part of the port type defintions
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<message name="SupplierRequestMessage">
    <part name="payload" element="client:SupplierProcessRequest"/>
</message>

<message name="SupplierResponseMessage">
    <part name="payload" element="client:SupplierProcessResponse"/>
</message>


<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
PORT TYPE DEFINITION - A port type groups a set of operations into
a logical service unit.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
<!-- portType implemented by the Supplier BPEL process -->
<portType name="ServicePT">
    <operation name="order">
        <input message="client:SupplierRequestMessage"/>
    </operation>
    <operation name="change">
        <input message="client:SupplierRequestMessage"/>
    </operation>
</portType>

<!-- portType implemented by the requester of Supplier BPEL process
for asynchronous callback purposes
-->
<portType name="ServicePTCallback">
    <operation name="orderResponse">
```

```
            <input message="client:SupplierResponseMessage"/>
        </operation>
        <operation name="orderChangeResponse">
            <input message="client:SupplierResponseMessage"/>
        </operation>
    </portType>

    <!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    PARTNER LINK TYPE DEFINITION
    the Supplier partnerLinkType binds the provider and
    requester portType into an asynchronous conversation.
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ -->
    <plnk:partnerLinkType name="Supplier">
        <plnk:role name="SupplierProvider">
            <plnk:portType name="client:ServicePT"/>
        </plnk:role>
        <plnk:role name="SupplierRequester">
            <plnk:portType name="client:ServicePTCallback"/>
        </plnk:role>
    </plnk:partnerLinkType>
</definitions>
```

A.2 Abstract BPEL

```
<!--
  ////////////////////////////////////////////////////////////////////////
  // Oracle JDeveloper BPEL Designer
  //
  // Created: Mon Sep 19 11:39:21 CEST 2005
  // Author:  hverbeek
  // Purpose: Asynchronous BPEL Process
  ////////////////////////////////////////////////////////////////////////
-->
<process
    name="Supplier"
    targetNamespace="http://services.qut.com/Supplier"
    xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:client="http://services.qut.com/Supplier"
    xmlns:ns1="http://www.w3.org/2001/XMLSchema"
    xmlns:ns2="tns"
    xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
>
    <!--
        List of services participating in this BPEL process
    -->
    <partnerLinks>
        <!--
            The 'client' role represents the requester of this service. It
```

```
        is used for callback. The location and correlation information
        associated with the client role are automatically set using
        WS-Addressing.
    -->
    <partnerLink
        name="client"
        partnerLinkType="client:Supplier"
        myRole="SupplierProvider"
        partnerRole="SupplierRequester"
    />
</partnerLinks>
<!--
    List of messages and XML documents used within this BPEL process
-->
<variables>
    <variable
        name="inputVariable"
        messageType="client:SupplierRequestMessage"
    />
    <variable
        name="outputVariable"
        messageType="client:SupplierResponseMessage"
    />
</variables>
<!--
    ORCHESTRATION LOGIC
-->
<sequence
    name="main"
>
    <receive
        name="order"
        partnerLink="client"
        portType="client:ServicePT"
        operation="order"
        variable="inputVariable"
        createInstance="yes"
    />
        <invoke
            name="orderResponse"
            partnerLink="client"
            portType="client:ServicePTCallback"
            operation="orderResponse"
            inputVariable="outputVariable"
        />
        <scope name="cancellationScope">
            <faultHandlers>
                <catch faultName="ns2:orderChange">
                <sequence name="change">
                    <invoke
```

```
                        name="orderChangeResponse"
                        partnerLink="client"
                        portType="client:ServicePTCallback"
                        operation="orderChangeResponse"
                        inputVariable="outputVariable"
                    />
                    <while>
                        <invoke
                            name="orderChangeResponse"
                            partnerLink="client"
                            portType="client:ServicePTCallback"
                            operation="orderChangeResponse"
                            inputVariable="outputVariable"
                        />
                    </while>
                </sequence>
            </catch>
        </faultHandlers>
        <eventHandlers>
            <onMessage p
                ortType="client:ServicePT"
                operation="change"
                variable="inputVariable"
                partnerLink="client"
            >
                <throw
                    name="throwFault"
                    faultName="ns2:orderChange"
                />
            </onMessage>
        </eventHandlers>
        <while>
            <invoke
                name="orderResponse"
                partnerLink="client"
                portType="client:ServicePTCallback"
                operation="orderResponse"
                inputVariable="outputVariable"/>
        </while>
    </scope>
</sequence>
</process>
```

A.3 Executable BPEL

```
<!--

////////////////////////////////////////////////////////////////////////
//////////////////
  // Oracle JDeveloper BPEL Designer
```

```
  //
  // Created: Mon Sep 19 11:39:21 CEST 2005
  // Author:  hverbeek
  // Purpose: Asynchronous BPEL Process

////////////////////////////////////////////////////////////////////////
//////////////////
-->
<process name="Supplier"
targetNamespace="http://services.qut.com/Supplier"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.service
s.functions.Xpath20"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:client="http://services.qut.com/Supplier"
xmlns:ora="http://schemas.oracle.com/xpath/extension"
xmlns:identityservice="http://xmlns.oracle.com/pcbpel/identityservice/loca
l" xmlns:ns1="http://www.w3.org/2001/XMLSchema"
xmlns:taskmngr="http://xmlns.oracle.com/pcbpel/taskservice/taskmanager"
xmlns:taskroutingservice="http://xmlns.oracle.com/pcbpel/taskservice/taskr
outingservice" xmlns:ns2="tns"
xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns:task="http://xmlns.oracle.com/pcbpel/taskservice/task"
xmlns:taskactionhandler="http://xmlns.oracle.com/pcbpel/taskservice/taskAc
tionHandler"
xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.service
s.functions.ExtFunc">
  <!-- ================================================================ -
->
  <!-- PARTNERLINKS                                                     -
->
  <!-- List of services participating in this BPEL process              -
->
  <!-- ================================================================ -
->
  <partnerLinks>
    <!--
  The 'client' role represents the requester of this service. It is
  used for callback. The location and correlation information associated
  with the client role are automatically set using WS-Addressing.
  -->
    <partnerLink name="client" partnerLinkType="client:Supplier"
myRole="SupplierProvider" partnerRole="SupplierRequester"/>
    <partnerLink myRole="TaskManagerCallbackListener"
name="TaskManagerService" partnerRole="TaskManager"
partnerLinkType="taskmngr:TaskManager"/>
    <partnerLink name="TaskRoutingService"
partnerRole="TaskRoutingService"
```

```
partnerLinkType="taskroutingservice:TaskRoutingService"/>
    <partnerLink myRole="HandleTaskActionRequester"
name="TaskActionHandler" partnerRole="HandleTaskActionProvider"
partnerLinkType="taskactionhandler:TaskActionHandler"/>
    <partnerLink name="IdentityService"
partnerRole="IdentityServiceProvider"
partnerLinkType="identityservice:IdentityService"/>
  </partnerLinks>
  <!-- ================================================================= -
->
  <!-- VARIABLES                                                         -
->
  <!-- List of messages and XML documents used within this BPEL process  -
->
  <!-- ================================================================= -
->
  <variables>
    <!-- Reference to the message passed as input during initiation -->
    <!-- Reference to the message that will be sent back to the
  requester during callback
  -->
    <variable name="inputVariable"
messageType="client:SupplierRequestMessage"/>
    <variable name="outputVariable"
messageType="client:SupplierResponseMessage"/>
    <variable name="nofItems" type="ns1:integer"/>
    <variable name="checkTaskVar" element="task:task"/>
    <variable name="checkChangedOrderVar" element="task:task"/>
  </variables>
  <!-- ================================================================= -
->
  <!-- ORCHESTRATION LOGIC                                               -
->
  <!-- Set of activities coordinating the flow of messages across the    -
->
  <!-- services integrated within this business process                  -
->
  <!-- ================================================================= -
->
  <sequence name="main">
    <!-- Receive input from requestor.
  Note: This maps to operation defined in Supplier.wsdl
  -->
    <!-- Asynchronous callback to the requester.
   Note: the callback location and correlation id is transparently handled
   using WS-addressing.
   -->
    <receive name="order" partnerLink="client" portType="client:ServicePT"
operation="order" variable="inputVariable" createInstance="yes"/>
    <sequence name="initOrder">
```

```
      <assign name="setNofItems">
        <copy>
          <from variable="inputVariable" part="payload"
query="/client:SupplierProcessRequest/client:nofItems"/>
          <to variable="nofItems"/>
        </copy>
      </assign>
    </sequence>
    <sequence name="preOrderResponse">
      <assign name="setOutput">
        <copy>
          <from variable="nofItems"/>
          <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse/client:nofItems"/>
        </copy>
      </assign>
      <scope name="checkTask" variableAccessSerializable="no"
xmlns:taskactionhandler="http://xmlns.oracle.com/pcbpel/taskservice/taskAc
tionHandler" xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:taskmngr="http://xmlns.oracle.com/pcbpel/taskservice/taskmanager"
xmlns:task="http://xmlns.oracle.com/pcbpel/taskservice/task"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ora="http://schemas.oracle.com/xpath/extension"
xmlns:wf="http://schemas.oracle.com/bpel/extension/workflow"
wf:key="checkTaskVar;taskConfigcheckTask.xml;SimpleUserActivity;&lt;%conca
t(string('Check order: '), string(bpws:getVariableData('nofItems')),
string(' items to go'))%&gt;;;bpws:getVariableData('outputVariable',
'payload', '/client:SupplierProcessResponse');;;;;">
        <variables>
          <variable name="oraBPMTaskMessage"
messageType="taskmngr:taskMessage"/>
          <variable name="oraBPMTaskErroredFaultMessage"
messageType="taskmngr:taskErroredMessage"/>
          <variable name="oraBPMTemporaryVariable" type="xsd:string"/>
        </variables>
        <correlationSets>
          <correlationSet name="oraBPMTaskIdCor"
properties="taskmngr:taskId"/>
        </correlationSets>
        <sequence>
          <assign name="setUserDefinedAttributes">
            <copy>
              <from expression="concat(string('Check order: '),
string(bpws:getVariableData('nofItems')), string(' items to go'))"/>
              <to variable="checkTaskVar" query="/task:task/task:title"/>
            </copy>
            <copy>
```

```
              <from expression="bpws:getVariableData('outputVariable',
'payload', '/client:SupplierProcessResponse')"/>
              <to variable="checkTaskVar"
query="/task:task/task:payload"/>
            </copy>
            <copy>
              <from expression="string('hverbeek')"/>
              <to variable="checkTaskVar"
query="/task:task/task:assigneeUsers[1]"/>
            </copy>
            <copy>
              <from expression="concat(ora:getProcessURL(),
string('/taskConfigcheckTask.xml'))"/>
              <to variable="checkTaskVar"
query="/task:task/task:taskType"/>
            </copy>
          </assign>
          <assign name="setSystemDefinedAttributes">
            <copy>
              <from expression="ora:getInstanceId()"/>
              <to variable="checkTaskVar"
query="/task:task/task:instanceId"/>
            </copy>
            <copy>
              <from expression="ora:getProcessId()"/>
              <to variable="checkTaskVar"
query="/task:task/task:processName"/>
            </copy>
            <copy>
              <from expression="ora:getProcessId()"/>
              <to variable="checkTaskVar"
query="/task:task/task:processId"/>
            </copy>
            <copy>
              <from expression="ora:getProcessVersion()"/>
              <to variable="checkTaskVar"
query="/task:task/task:processVersion"/>
            </copy>
            <copy>
              <from expression="ora:getDomainId()"/>
              <to variable="checkTaskVar"
query="/task:task/task:domainId"/>
            </copy>
            <copy>
              <from expression="ora:getProcessOwnerId()"/>
              <to variable="checkTaskVar"
query="/task:task/task:processOwner"/>
            </copy>
            <copy>
              <from expression="string('SINGLE_APPROVAL')"/>
```

```
              <to variable="checkTaskVar"
query="/task:task/task:pattern"/>
            </copy>
            <copy>
              <from expression="false()"/>
              <to variable="checkTaskVar"
query="/task:task/task:hasSubTasks"/>
            </copy>
            <copy>
              <from variable="checkTaskVar"/>
              <to variable="oraBPMTaskMessage" part="payload"/>
            </copy>
          </assign>
          <scope name="initiateTask">
            <faultHandlers>
              <catch faultName="taskmngr:taskErroredFault"
faultVariable="oraBPMTaskErroredFaultMessage">
                <assign name="readErroredTask">
                  <copy>
                    <from variable="oraBPMTaskErroredFaultMessage"
part="payload"/>
                    <to variable="oraBPMTaskMessage" part="payload"/>
                  </copy>
                </assign>
              </catch>
            </faultHandlers>
            <sequence>
              <invoke name="initiateTask" partnerLink="TaskManagerService"
portType="taskmngr:TaskManager" operation="initiateTask"
inputVariable="oraBPMTaskMessage" outputVariable="oraBPMTaskMessage"/>
            </sequence>
          </scope>
          <sequence>
            <invoke name="initiateTaskActionHandler"
partnerLink="TaskActionHandler"
portType="taskactionhandler:TaskActionHandler" operation="initiate"
inputVariable="oraBPMTaskMessage">
              <correlations>
                <correlation set="oraBPMTaskIdCor" initiate="yes"
pattern="out"/>
              </correlations>
            </invoke>
            <receive name="receiveUpdatedTask"
partnerLink="TaskActionHandler"
portType="taskactionhandler:TaskActionHandlerCallback"
operation="onTaskCompleted" variable="oraBPMTaskMessage"
createInstance="no">
              <correlations>
                <correlation set="oraBPMTaskIdCor" initiate="no"/>
              </correlations>
```

```
          </receive>
        </sequence>
        <assign name="readUpdatedTask">
          <copy>
            <from variable="oraBPMTaskMessage" part="payload"/>
            <to variable="checkTaskVar"/>
          </copy>
        </assign>
      </sequence>
    </scope>
    <switch name="taskSwitch"
xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:tt="http://xmlns.oracle.com/pcbpel/taskservice/tasktype"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <case condition="bpws:getVariableData('checkTaskVar',
'/task:task/task:state') = 'COMPLETED' and
bpws:getVariableData('checkTaskVar',
'/task:task/task:conclusion') = 'ACCEPT'">
        <bpelx:annotation>
          <bpelx:pattern>Task outcome is ACCEPT
    </bpelx:pattern>
        </bpelx:annotation>
        <sequence>
          <assign name="copyPayloadFromTask">
            <copy>
              <from variable="checkTaskVar"
query="/task:task/task:payload"/>
              <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse"/>
            </copy>
          </assign>
        </sequence>
      </case>
      <case condition="bpws:getVariableData('checkTaskVar',
'/task:task/task:state') = 'COMPLETED' and
bpws:getVariableData('checkTaskVar',
'/task:task/task:conclusion') = 'REJECT'">
        <bpelx:annotation>
          <bpelx:pattern>Task outcome is REJECT
    </bpelx:pattern>
        </bpelx:annotation>
        <sequence>
          <assign name="copyPayloadFromTask">
            <copy>
              <from variable="checkTaskVar"
query="/task:task/task:payload"/>
              <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse"/>
```

```
            </copy>
          </assign>
        </sequence>
      </case>
      <otherwise>
        <bpelx:annotation>
          <bpelx:pattern>Task is EXPIRED, WITHDRAWN or ERRORED
    </bpelx:pattern>
        </bpelx:annotation>
        <sequence>
          <assign name="copyPayloadFromTask">
            <copy>
              <from variable="checkTaskVar"
query="/task:task/task:payload"/>
              <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse"/>
            </copy>
          </assign>
        </sequence>
      </otherwise>
    </switch>
  </sequence>
  <invoke name="orderResponse" partnerLink="client"
portType="client:ServicePTCallback" operation="orderResponse"
inputVariable="outputVariable"/>
  <sequence name="postOrderResponse">
    <assign name="decreaseNofItems">
      <copy>
        <from expression="bpws:getVariableData('nofItems') -
bpws:getVariableData('outputVariable','payload','/client:SupplierProcessRe
sponse/client:nofItems')"/>
        <to variable="nofItems"/>
      </copy>
    </assign>
  </sequence>
  <scope name="cancellationScope">
    <faultHandlers>
      <catch faultName="ns2:orderChange">
        <sequence name="orderChangedSequence">
          <sequence name="initChangeOrder">
            <assign name="setChangedNofItems">
              <copy>
                <from variable="inputVariable" part="payload"
query="/client:SupplierProcessRequest/client:nofItems"/>
                <to variable="nofItems"/>
              </copy>
            </assign>
          </sequence>
          <sequence name="orderChangedResponseSequence">
            <sequence name="preChangeOrderResponse">
```

```
                <assign name="setChangedOutput">
                  <copy>
                    <from variable="nofItems"/>
                    <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse/client:nofItems"/>
                  </copy>
                </assign>
                <scope name="checkChangedOrder"
variableAccessSerializable="no"
xmlns:taskactionhandler="http://xmlns.oracle.com/pcbpel/taskservice/taskAc
tionHandler" xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:taskmngr="http://xmlns.oracle.com/pcbpel/taskservice/taskmanager"
xmlns:task="http://xmlns.oracle.com/pcbpel/taskservice/task"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ora="http://schemas.oracle.com/xpath/extension"
xmlns:wf="http://schemas.oracle.com/bpel/extension/workflow"
wf:key="checkChangedOrderVar;taskConfigcheckChangedOrder.xml;SimpleUserAct
ivity;&lt;%concat(string('Check Changed Order: '),
string(bpws:getVariableData('nofItems')), string(' items to
go'))%&gt;;bpws:getVariableData('outputVariable', 'payload',
'/client:SupplierProcessResponse');;;;;">
                  <variables>
                    <variable name="oraBPMTaskMessage"
messageType="taskmngr:taskMessage"/>
                    <variable name="oraBPMTaskErroredFaultMessage"
messageType="taskmngr:taskErroredMessage"/>
                    <variable name="oraBPMTemporaryVariable"
type="xsd:string"/>
                  </variables>
                  <correlationSets>
                    <correlationSet name="oraBPMTaskIdCor"
properties="taskmngr:taskId"/>
                  </correlationSets>
                  <sequence>
                    <assign name="setUserDefinedAttributes">
                      <copy>
                        <from expression="concat(string('Check Changed
Order: '), string(bpws:getVariableData('nofItems')), string(' items to
go'))"/>
                        <to variable="checkChangedOrderVar"
query="/task:task/task:title"/>
                      </copy>
                      <copy>
                        <from
expression="bpws:getVariableData('outputVariable', 'payload',
'/client:SupplierProcessResponse')"/>
                        <to variable="checkChangedOrderVar"
```

```
query="/task:task/task:payload"/>
                      </copy>
                      <copy>
                        <from expression="string('hverbeek')"/>
                        <to variable="checkChangedOrderVar"
query="/task:task/task:assigneeUsers[1]"/>
                      </copy>
                      <copy>
                        <from expression="concat(ora:getProcessURL(),
string('/taskConfigcheckChangedOrder.xml'))"/>
                        <to variable="checkChangedOrderVar"
query="/task:task/task:taskType"/>
                      </copy>
                    </assign>
                    <assign name="setSystemDefinedAttributes">
                      <copy>
                        <from expression="ora:getInstanceId()"/>
                        <to variable="checkChangedOrderVar"
query="/task:task/task:instanceId"/>
                      </copy>
                      <copy>
                        <from expression="ora:getProcessId()"/>
                        <to variable="checkChangedOrderVar"
query="/task:task/task:processName"/>
                      </copy>
                      <copy>
                        <from expression="ora:getProcessId()"/>
                        <to variable="checkChangedOrderVar"
query="/task:task/task:processId"/>
                      </copy>
                      <copy>
                        <from expression="ora:getProcessVersion()"/>
                        <to variable="checkChangedOrderVar"
query="/task:task/task:processVersion"/>
                      </copy>
                      <copy>
                        <from expression="ora:getDomainId()"/>
                        <to variable="checkChangedOrderVar"
query="/task:task/task:domainId"/>
                      </copy>
                      <copy>
                        <from expression="ora:getProcessOwnerId()"/>
                        <to variable="checkChangedOrderVar"
query="/task:task/task:processOwner"/>
                      </copy>
                      <copy>
                        <from expression="string('SINGLE_APPROVAL')"/>
                        <to variable="checkChangedOrderVar"
query="/task:task/task:pattern"/>
                      </copy>
```

```
                    <copy>
                      <from expression="false()"/>
                      <to variable="checkChangedOrderVar"
query="/task:task/task:hasSubTasks"/>
                    </copy>
                    <copy>
                      <from variable="checkChangedOrderVar"/>
                      <to variable="oraBPMTaskMessage" part="payload"/>
                    </copy>
                  </assign>
                  <scope name="initiateTask">
                    <faultHandlers>
                      <catch faultName="taskmngr:taskErroredFault"
faultVariable="oraBPMTaskErroredFaultMessage">
                        <assign name="readErroredTask">
                          <copy>
                            <from
variable="oraBPMTaskErroredFaultMessage" part="payload"/>
                              <to variable="oraBPMTaskMessage"
part="payload"/>
                          </copy>
                        </assign>
                      </catch>
                    </faultHandlers>
                    <sequence>
                      <invoke name="initiateTask"
partnerLink="TaskManagerService" portType="taskmngr:TaskManager"
operation="initiateTask" inputVariable="oraBPMTaskMessage"
outputVariable="oraBPMTaskMessage"/>
                    </sequence>
                  </scope>
                  <sequence>
                    <invoke name="initiateTaskActionHandler"
partnerLink="TaskActionHandler"
portType="taskactionhandler:TaskActionHandler" operation="initiate"
inputVariable="oraBPMTaskMessage">
                      <correlations>
                        <correlation set="oraBPMTaskIdCor"
initiate="yes" pattern="out"/>
                      </correlations>
                    </invoke>
                    <receive name="receiveUpdatedTask"
partnerLink="TaskActionHandler"
portType="taskactionhandler:TaskActionHandlerCallback"
operation="onTaskCompleted" variable="oraBPMTaskMessage"
createInstance="no">
                      <correlations>
                        <correlation set="oraBPMTaskIdCor"
initiate="no"/>
                      </correlations>
```

```
                    </receive>
                  </sequence>
                  <assign name="readUpdatedTask">
                    <copy>
                      <from variable="oraBPMTaskMessage"
part="payload"/>
                      <to variable="checkChangedOrderVar"/>
                    </copy>
                  </assign>
                </sequence>
              </scope>
              <switch name="taskSwitch"
xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:tt="http://xmlns.oracle.com/pcbpel/taskservice/tasktype"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
                <case
condition="bpws:getVariableData('checkChangedOrderVar',
'/task:task/task:state') = 'COMPLETED' and
bpws:getVariableData('checkChangedOrderVar',
'/task:task/task:conclusion') = 'ACCEPT'">
                  <bpelx:annotation>
                    <bpelx:pattern>Task outcome is ACCEPT
          </bpelx:pattern>
                  </bpelx:annotation>
                  <sequence>
                    <assign name="copyPayloadFromTask">
                      <copy>
                        <from variable="checkChangedOrderVar"
query="/task:task/task:payload"/>
                        <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse"/>
                      </copy>
                    </assign>
                  </sequence>
                </case>
                <case
condition="bpws:getVariableData('checkChangedOrderVar',
'/task:task/task:state') = 'COMPLETED' and
bpws:getVariableData('checkChangedOrderVar',
'/task:task/task:conclusion') = 'REJECT'">
                  <bpelx:annotation>
                    <bpelx:pattern>Task outcome is REJECT
          </bpelx:pattern>
                  </bpelx:annotation>
                  <sequence>
                    <assign name="copyPayloadFromTask">
                      <copy>
                        <from variable="checkChangedOrderVar"
```

```
query="/task:task/task:payload"/>
                        <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse"/>
                      </copy>
                    </assign>
                  </sequence>
                </case>
                <otherwise>
                  <bpelx:annotation>
                    <bpelx:pattern>Task is EXPIRED, WITHDRAWN or ERRORED
            </bpelx:pattern>
                  </bpelx:annotation>
                  <sequence>
                    <assign name="copyPayloadFromTask">
                      <copy>
                        <from variable="checkChangedOrderVar"
query="/task:task/task:payload"/>
                        <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse"/>
                      </copy>
                    </assign>
                  </sequence>
                </otherwise>
              </switch>
            </sequence>
            <invoke name="orderChangeResponse" partnerLink="client"
portType="client:ServicePTCallback" operation="orderChangeResponse"
inputVariable="outputVariable"/>
            <sequence name="postChangeOrderResponse">
              <assign name="decreaseChangedNofItems">
                <copy>
                  <from expression="bpws:getVariableData('nofItems') -
bpws:getVariableData('outputVariable','payload','/client:SupplierProcessRe
sponse/client:nofItems')"/>
                  <to variable="nofItems"/>
                </copy>
              </assign>
            </sequence>
          </sequence>
          <while name="remainingOrderChangedItems"
condition="bpws:getVariableData('nofItems') &gt; 0">
            <sequence name="orderChangedResponseSequence">
              <sequence name="preChangeOrderResponse">
                <assign name="setChangedOutput">
                  <copy>
                    <from variable="nofItems"/>
                    <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse/client:nofItems"/>
                  </copy>
                </assign>
```

```
                <scope name="checkChangedOrder"
variableAccessSerializable="no"
xmlns:taskactionhandler="http://xmlns.oracle.com/pcbpel/taskservice/taskAc
tionHandler" xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:taskmngr="http://xmlns.oracle.com/pcbpel/taskservice/taskmanager"
xmlns:task="http://xmlns.oracle.com/pcbpel/taskservice/task"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ora="http://schemas.oracle.com/xpath/extension"
xmlns:wf="http://schemas.oracle.com/bpel/extension/workflow"
wf:key="checkChangedOrderVar;taskConfigcheckChangedOrder.xml;SimpleUserAct
ivity;&lt;%concat(string('Check Changed Order: '),
string(bpws:getVariableData('nofItems')), string(' items to
go'))%&gt;;;bpws:getVariableData('outputVariable', 'payload',
'/client:SupplierProcessResponse');;;;;">
                    <variables>
                      <variable name="oraBPMTaskMessage"
messageType="taskmngr:taskMessage"/>
                      <variable name="oraBPMTaskErroredFaultMessage"
messageType="taskmngr:taskErroredMessage"/>
                      <variable name="oraBPMTemporaryVariable"
type="xsd:string"/>
                    </variables>
                    <correlationSets>
                      <correlationSet name="oraBPMTaskIdCor"
properties="taskmngr:taskId"/>
                    </correlationSets>
                    <sequence>
                      <assign name="setUserDefinedAttributes">
                        <copy>
                          <from expression="concat(string('Check Changed
Order: '), string(bpws:getVariableData('nofItems')), string(' items to
go'))"/>
                          <to variable="checkChangedOrderVar"
query="/task:task/task:title"/>
                        </copy>
                        <copy>
                          <from
expression="bpws:getVariableData('outputVariable', 'payload',
'/client:SupplierProcessResponse')"/>
                          <to variable="checkChangedOrderVar"
query="/task:task/task:payload"/>
                        </copy>
                        <copy>
                          <from expression="string('hverbeek')"/>
                          <to variable="checkChangedOrderVar"
query="/task:task/task:assigneeUsers[1]"/>
                        </copy>
```

```
                        <copy>
                          <from expression="concat(ora:getProcessURL(),
string('/taskConfigcheckChangedOrder.xml'))"/>
                          <to variable="checkChangedOrderVar"
query="/task:task/task:taskType"/>
                        </copy>
                      </assign>
                      <assign name="setSystemDefinedAttributes">
                        <copy>
                          <from expression="ora:getInstanceId()"/>
                          <to variable="checkChangedOrderVar"
query="/task:task/task:instanceId"/>
                        </copy>
                        <copy>
                          <from expression="ora:getProcessId()"/>
                          <to variable="checkChangedOrderVar"
query="/task:task/task:processName"/>
                        </copy>
                        <copy>
                          <from expression="ora:getProcessId()"/>
                          <to variable="checkChangedOrderVar"
query="/task:task/task:processId"/>
                        </copy>
                        <copy>
                          <from expression="ora:getProcessVersion()"/>
                          <to variable="checkChangedOrderVar"
query="/task:task/task:processVersion"/>
                        </copy>
                        <copy>
                          <from expression="ora:getDomainId()"/>
                          <to variable="checkChangedOrderVar"
query="/task:task/task:domainId"/>
                        </copy>
                        <copy>
                          <from expression="ora:getProcessOwnerId()"/>
                          <to variable="checkChangedOrderVar"
query="/task:task/task:processOwner"/>
                        </copy>
                        <copy>
                          <from expression="string('SINGLE_APPROVAL')"/>
                          <to variable="checkChangedOrderVar"
query="/task:task/task:pattern"/>
                        </copy>
                        <copy>
                          <from expression="false()"/>
                          <to variable="checkChangedOrderVar"
query="/task:task/task:hasSubTasks"/>
                        </copy>
                        <copy>
                          <from variable="checkChangedOrderVar"/>
```

```
                        <to variable="oraBPMTaskMessage"
part="payload"/>
                      </copy>
                    </assign>
                    <scope name="initiateTask">
                      <faultHandlers>
                        <catch faultName="taskmngr:taskErroredFault"
faultVariable="oraBPMTaskErroredFaultMessage">
                          <assign name="readErroredTask">
                            <copy>
                              <from
variable="oraBPMTaskErroredFaultMessage" part="payload"/>
                              <to variable="oraBPMTaskMessage"
part="payload"/>
                            </copy>
                          </assign>
                        </catch>
                      </faultHandlers>
                      <sequence>
                        <invoke name="initiateTask"
partnerLink="TaskManagerService" portType="taskmngr:TaskManager"
operation="initiateTask" inputVariable="oraBPMTaskMessage"
outputVariable="oraBPMTaskMessage"/>
                      </sequence>
                    </scope>
                    <sequence>
                      <invoke name="initiateTaskActionHandler"
partnerLink="TaskActionHandler"
portType="taskactionhandler:TaskActionHandler" operation="initiate"
inputVariable="oraBPMTaskMessage">
                        <correlations>
                          <correlation set="oraBPMTaskIdCor"
initiate="yes" pattern="out"/>
                        </correlations>
                      </invoke>
                      <receive name="receiveUpdatedTask"
partnerLink="TaskActionHandler"
portType="taskactionhandler:TaskActionHandlerCallback"
operation="onTaskCompleted" variable="oraBPMTaskMessage"
createInstance="no">
                        <correlations>
                          <correlation set="oraBPMTaskIdCor"
initiate="no"/>
                        </correlations>
                      </receive>
                    </sequence>
                    <assign name="readUpdatedTask">
                      <copy>
                        <from variable="oraBPMTaskMessage"
part="payload"/>
```

```
                        <to variable="checkChangedOrderVar"/>
                      </copy>
                    </assign>
                  </sequence>
                </scope>
                <switch name="taskSwitch"
xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:tt="http://xmlns.oracle.com/pcbpel/taskservice/tasktype"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
                  <case
condition="bpws:getVariableData('checkChangedOrderVar',
'/task:task/task:state') = 'COMPLETED' and
bpws:getVariableData('checkChangedOrderVar',
'/task:task/task:conclusion') = 'ACCEPT'">
                    <bpelx:annotation>
                      <bpelx:pattern>Task outcome is ACCEPT
            </bpelx:pattern>
                    </bpelx:annotation>
                    <sequence>
                      <assign name="copyPayloadFromTask">
                        <copy>
                          <from variable="checkChangedOrderVar"
query="/task:task/task:payload"/>
                          <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse"/>
                        </copy>
                      </assign>
                    </sequence>
                  </case>
                  <case
condition="bpws:getVariableData('checkChangedOrderVar',
'/task:task/task:state') = 'COMPLETED' and
bpws:getVariableData('checkChangedOrderVar',
'/task:task/task:conclusion') = 'REJECT'">
                    <bpelx:annotation>
                      <bpelx:pattern>Task outcome is REJECT
            </bpelx:pattern>
                    </bpelx:annotation>
                    <sequence>
                      <assign name="copyPayloadFromTask">
                        <copy>
                          <from variable="checkChangedOrderVar"
query="/task:task/task:payload"/>
                          <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse"/>
                        </copy>
                      </assign>
                    </sequence>
```

```
                    </case>
                    <otherwise>
                      <bpelx:annotation>
                        <bpelx:pattern>Task is EXPIRED, WITHDRAWN or
ERRORED
            </bpelx:pattern>
                      </bpelx:annotation>
                      <sequence>
                        <assign name="copyPayloadFromTask">
                          <copy>
                            <from variable="checkChangedOrderVar"
query="/task:task/task:payload"/>
                            <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse"/>
                          </copy>
                        </assign>
                      </sequence>
                    </otherwise>
                  </switch>
                </sequence>
                <invoke name="orderChangeResponse" partnerLink="client"
portType="client:ServicePTCallback" operation="orderChangeResponse"
inputVariable="outputVariable"/>
                <sequence name="postChangeOrderResponse">
                  <assign name="decreaseChangedNofItems">
                    <copy>
                      <from expression="bpws:getVariableData('nofItems') -
bpws:getVariableData('outputVariable','payload','/client:SupplierProcessRe
sponse/client:nofItems')"/>
                      <to variable="nofItems"/>
                    </copy>
                  </assign>
                </sequence>
              </sequence>
            </while>
          </sequence>
        </catch>
      </faultHandlers>
      <eventHandlers>
        <onMessage portType="client:ServicePT" operation="change"
variable="inputVariable" partnerLink="client">
          <throw name="throwFault" faultName="ns2:orderChange"/>
        </onMessage>
      </eventHandlers>
      <sequence name="orderSequence">
        <while name="remainingOrderItems"
condition="bpws:getVariableData('nofItems') &gt; 0">
          <sequence name="orderResponseSequence">
            <sequence name="preOrderResponse">
              <assign name="setOutput">
```

```
                <copy>
                  <from variable="nofItems"/>
                  <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse/client:nofItems"/>
                </copy>
              </assign>
              <scope name="checkTask" variableAccessSerializable="no"
xmlns:taskactionhandler="http://xmlns.oracle.com/pcbpel/taskservice/taskAc
tionHandler" xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:taskmngr="http://xmlns.oracle.com/pcbpel/taskservice/taskmanager"
xmlns:task="http://xmlns.oracle.com/pcbpel/taskservice/task"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ora="http://schemas.oracle.com/xpath/extension"
xmlns:wf="http://schemas.oracle.com/bpel/extension/workflow"
wf:key="checkTaskVar;taskConfigcheckTask.xml;SimpleUserActivity;&lt;%conca
t(string('Check order: '), string(bpws:getVariableData('nofItems')),
string(' items to go'))%&gt;;bpws:getVariableData('outputVariable',
'payload', '/client:SupplierProcessResponse');;;;;">
                <variables>
                  <variable name="oraBPMTaskMessage"
messageType="taskmngr:taskMessage"/>
                  <variable name="oraBPMTaskErroredFaultMessage"
messageType="taskmngr:taskErroredMessage"/>
                  <variable name="oraBPMTemporaryVariable"
type="xsd:string"/>
                </variables>
                <correlationSets>
                  <correlationSet name="oraBPMTaskIdCor"
properties="taskmngr:taskId"/>
                </correlationSets>
                <sequence>
                  <assign name="setUserDefinedAttributes">
                    <copy>
                      <from expression="concat(string('Check order: '),
string(bpws:getVariableData('nofItems')), string(' items to go'))"/>
                      <to variable="checkTaskVar"
query="/task:task/task:title"/>
                    </copy>
                    <copy>
                      <from
expression="bpws:getVariableData('outputVariable', 'payload',
'/client:SupplierProcessResponse')"/>
                      <to variable="checkTaskVar"
query="/task:task/task:payload"/>
                    </copy>
                    <copy>
                      <from expression="string('hverbeek')"/>
```

```
                      <to variable="checkTaskVar"
query="/task:task/task:assigneeUsers[1]"/>
                    </copy>
                    <copy>
                      <from expression="concat(ora:getProcessURL(),
string('/taskConfigcheckTask.xml'))"/>
                      <to variable="checkTaskVar"
query="/task:task/task:taskType"/>
                    </copy>
                  </assign>
                  <assign name="setSystemDefinedAttributes">
                    <copy>
                      <from expression="ora:getInstanceId()"/>
                      <to variable="checkTaskVar"
query="/task:task/task:instanceId"/>
                    </copy>
                    <copy>
                      <from expression="ora:getProcessId()"/>
                      <to variable="checkTaskVar"
query="/task:task/task:processName"/>
                    </copy>
                    <copy>
                      <from expression="ora:getProcessId()"/>
                      <to variable="checkTaskVar"
query="/task:task/task:processId"/>
                    </copy>
                    <copy>
                      <from expression="ora:getProcessVersion()"/>
                      <to variable="checkTaskVar"
query="/task:task/task:processVersion"/>
                    </copy>
                    <copy>
                      <from expression="ora:getDomainId()"/>
                      <to variable="checkTaskVar"
query="/task:task/task:domainId"/>
                    </copy>
                    <copy>
                      <from expression="ora:getProcessOwnerId()"/>
                      <to variable="checkTaskVar"
query="/task:task/task:processOwner"/>
                    </copy>
                    <copy>
                      <from expression="string('SINGLE_APPROVAL')"/>
                      <to variable="checkTaskVar"
query="/task:task/task:pattern"/>
                    </copy>
                    <copy>
                      <from expression="false()"/>
                      <to variable="checkTaskVar"
query="/task:task/task:hasSubTasks"/>
```

```
                    </copy>
                    <copy>
                      <from variable="checkTaskVar"/>
                      <to variable="oraBPMTaskMessage" part="payload"/>
                    </copy>
                  </assign>
                  <scope name="initiateTask">
                    <faultHandlers>
                      <catch faultName="taskmngr:taskErroredFault"
faultVariable="oraBPMTaskErroredFaultMessage">
                        <assign name="readErroredTask">
                          <copy>
                            <from variable="oraBPMTaskErroredFaultMessage"
part="payload"/>
                            <to variable="oraBPMTaskMessage"
part="payload"/>
                          </copy>
                        </assign>
                      </catch>
                    </faultHandlers>
                    <sequence>
                      <invoke name="initiateTask"
partnerLink="TaskManagerService" portType="taskmngr:TaskManager"
operation="initiateTask" inputVariable="oraBPMTaskMessage"
outputVariable="oraBPMTaskMessage"/>
                    </sequence>
                  </scope>
                  <sequence>
                    <invoke name="initiateTaskActionHandler"
partnerLink="TaskActionHandler"
portType="taskactionhandler:TaskActionHandler" operation="initiate"
inputVariable="oraBPMTaskMessage">
                      <correlations>
                        <correlation set="oraBPMTaskIdCor" initiate="yes"
pattern="out"/>
                      </correlations>
                    </invoke>
                    <receive name="receiveUpdatedTask"
partnerLink="TaskActionHandler"
portType="taskactionhandler:TaskActionHandlerCallback"
operation="onTaskCompleted" variable="oraBPMTaskMessage"
createInstance="no">
                      <correlations>
                        <correlation set="oraBPMTaskIdCor" initiate="no"/>
                      </correlations>
                    </receive>
                  </sequence>
                  <assign name="readUpdatedTask">
                    <copy>
                      <from variable="oraBPMTaskMessage" part="payload"/>
```

```
                    <to variable="checkTaskVar"/>
                  </copy>
                </assign>
              </sequence>
            </scope>
            <switch name="taskSwitch"
xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:tt="http://xmlns.oracle.com/pcbpel/taskservice/tasktype"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
              <case condition="bpws:getVariableData('checkTaskVar',
'/task:task/task:state') = 'COMPLETED' and
bpws:getVariableData('checkTaskVar',
'/task:task/task:conclusion') = 'ACCEPT'">
                <bpelx:annotation>
                  <bpelx:pattern>Task outcome is ACCEPT
          </bpelx:pattern>
                </bpelx:annotation>
                <sequence>
                  <assign name="copyPayloadFromTask">
                    <copy>
                      <from variable="checkTaskVar"
query="/task:task/task:payload"/>
                      <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse"/>
                    </copy>
                  </assign>
                </sequence>
              </case>
              <case condition="bpws:getVariableData('checkTaskVar',
'/task:task/task:state') = 'COMPLETED' and
bpws:getVariableData('checkTaskVar',
'/task:task/task:conclusion') = 'REJECT'">
                <bpelx:annotation>
                  <bpelx:pattern>Task outcome is REJECT
          </bpelx:pattern>
                </bpelx:annotation>
                <sequence>
                  <assign name="copyPayloadFromTask">
                    <copy>
                      <from variable="checkTaskVar"
query="/task:task/task:payload"/>
                      <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse"/>
                    </copy>
                  </assign>
                </sequence>
              </case>
              <otherwise>
```

```
                <bpelx:annotation>
                  <bpelx:pattern>Task is EXPIRED, WITHDRAWN or ERRORED
          </bpelx:pattern>
                </bpelx:annotation>
                <sequence>
                  <assign name="copyPayloadFromTask">
                    <copy>
                      <from variable="checkTaskVar"
query="/task:task/task:payload"/>
                      <to variable="outputVariable" part="payload"
query="/client:SupplierProcessResponse"/>
                    </copy>
                  </assign>
                </sequence>
              </otherwise>
            </switch>
          </sequence>
          <invoke name="orderResponse" partnerLink="client"
portType="client:ServicePTCallback" operation="orderResponse"
inputVariable="outputVariable"/>
          <sequence name="postOrderResponse">
            <assign name="decreaseNofItems">
              <copy>
                <from expression="bpws:getVariableData('nofItems') -
bpws:getVariableData('outputVariable','payload','/client:SupplierProcessRe
sponse/client:nofItems')"/>
                <to variable="nofItems"/>
              </copy>
            </assign>
          </sequence>
        </sequence>
      </while>
    </sequence>
  </scope>
 </sequence>
</process>
```

B Actual SOAP message logs

This appendix contains two logs with SOAP messages. The first log contains the two messages received by the BPEL server running the Supplier process, whereas the second log contains the three messages received by the BPEL server running the Customer process. First, however, we detail how we can link Figure 17 to both logs. As Figure 17 and both logs listed here are in XML format, we use XPath-like expressions ("//e" means an e-element anywhere in the XML tree, and "e@a" means the a-attribute of the e-element).

//Process@id Value of //ServiceName@xmlns:snns
//Process@description Added manually

//ProcessInstance@id Value of //MessageID
//ProcessInstance@description "Instance " concatenated with value of //MessageId@orabpel:rootId
//WorkflowModelElement Value of SOAPAction
//EventType "complete"
//TimeStamp (Converted) timestamp of response (as the request does not contain any timestamp)

Consider for example the first message in Appendix B.1. The process identifier is `http://services.qut.com/Supplier`, the process instance identifier is `bpel://localhost/default/Customer~1.1/301-BpInv0-BpSeq0.3-3`, the WorkflowModelElement (i.e., the identifier pointing to the activity) is `order`, and the timestamp is `2005-10-20T11:54:09-00:00` (Thu, 20 Oct 2005 11:54:09 GMT). This corresponds to the following event in the event log:

```
...
  <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>
 ...
```

To improve the readability of the logs, we have inserted line breaks.

B.1 From the Customer process to the Supplier process

```
==============
Listen Port: 1234
Target Host: GA2494
Target Port: 9700
==== Request ====
POST /orabpel/default/Supplier HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/#axisVersion#
Host: GA2494:1234
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: "order"
```

```
Content-Length: 979
Connection: close

<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
    <soapenv:Header>
        <MessageID
            xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing"
            xmlns:orabpel="http://schemas.oracle.com/bpel"
            orabpel:rootId="301" orabpel:parentId="301"
            orabpel:priority="3"
        >
            bpel://localhost/default/Customer~1.1/301-BpInv0-BpSeq0.3-3
        </MessageID>
        <ReplyTo
            xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing"
        >
            <Address>
                http://GA2550:9710/orabpel/default/Customer/1.1/Supplier/S
upplierRequester
            </Address>
            <PortType
                xmlns:ptns="http://services.qut.com/Supplier"
            >
                ptns:ServicePTCallback
            </PortType>
            <ServiceName
                xmlns:snns="http://services.qut.com/Supplier"
            >
                snns:ServicePTCallbackService
            </ServiceName>
        </ReplyTo>
    </soapenv:Header>
    <soapenv:Body>
        <SupplierProcessRequest
            xmlns="http://services.qut.com/Supplier"
        >
            <nofItems>
                5
            </nofItems>
        </SupplierProcessRequest>
    </soapenv:Body>
</soapenv:Envelope>
==== Response ====
HTTP/1.1 200 OK
Date: Thu, 20 Oct 2005 11:54:09 GMT
Server: Oracle Application Server Containers for J2EE 10g (10.1.2.0.0)
```

```
Connection: Close
Content-Type: text/xml; charset=utf-8

<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
    <soapenv:Body/>
</soapenv:Envelope>
==============

==============
Listen Port: 1234
Target Host: GA2494
Target Port: 9700
==== Request ====
POST /orabpel/default/Supplier HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/#axisVersion#
Host: GA2494:1234
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: "change"
Content-Length: 507
Connection: close

<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
    <soapenv:Header>
        <RelatesTo
            xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing"
        >
            bpel://localhost/default/Customer~1.1/301-BpInv0-BpSeq0.3-3
        </RelatesTo>
    </soapenv:Header>
    <soapenv:Body>
        <SupplierProcessRequest
            xmlns="http://services.qut.com/Supplier"
        >
            <nofItems>
                5
            </nofItems>
        </SupplierProcessRequest>
    </soapenv:Body>
</soapenv:Envelope>
```

```
==== Response ====
HTTP/1.1 200 OK
Date: Thu, 20 Oct 2005 11:58:20 GMT
Server: Oracle Application Server Containers for J2EE 10g (10.1.2.0.0)
Connection: Close
Content-Type: text/xml; charset=utf-8

<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
    <soapenv:Body/>
</soapenv:Envelope>
==============
```

B.2 From the Supplier process to the client

```
==============
Listen Port: 9710
Target Host: GA2550
Target Port: 9700
==== Request ====
POST /orabpel/default/Customer/1.1/Supplier/SupplierRequester HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/#axisVersion#
Host: GA2550:9710
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: "orderResponse"
Content-Length: 509
Connection: close

<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
    <soapenv:Header>
        <RelatesTo
            xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing"
        >
            bpel://localhost/default/Customer~1.1/301-BpInv0-BpSeq0.3-3
        </RelatesTo>
    </soapenv:Header>
    <soapenv:Body>
        <SupplierProcessResponse
            xmlns="http://services.qut.com/Supplier"
```

```
        >
            <nofItems>
                1
            </nofItems>
        </SupplierProcessResponse>
    </soapenv:Body>
</soapenv:Envelope>
==== Response ====
HTTP/1.1 200 OK
Date: Thu, 20 Oct 2005 11:58:08 GMT
Server: Oracle Application Server Containers for J2EE 10g (10.1.2.0.0)
Connection: Close
Content-Type: text/xml; charset=utf-8

<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
    <soapenv:Body/>
</soapenv:Envelope>
==============

==============
Listen Port: 9710
Target Host: GA2550
Target Port: 9700
==== Request ====
POST /orabpel/default/Customer/1.1/Supplier/SupplierRequester HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/#axisVersion#
Host: GA2550:9710
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: "orderChangeResponse"
Content-Length: 509
Connection: close

<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
    <soapenv:Header>
        <RelatesTo
            xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing"
        >
            bpel://localhost/default/Customer~1.1/301-BpInv0-BpSeq0.3-3
        </RelatesTo>
```

```
    </soapenv:Header>
    <soapenv:Body>
        <SupplierProcessResponse
            xmlns="http://services.qut.com/Supplier"
        >
            <nofItems>
                2
            </nofItems>
        </SupplierProcessResponse>
    </soapenv:Body>
</soapenv:Envelope>
==== Response ====
HTTP/1.1 200 OK
Date: Thu, 20 Oct 2005 11:58:35 GMT
Server: Oracle Application Server Containers for J2EE 10g (10.1.2.0.0)
Connection: Close
Content-Type: text/xml; charset=utf-8

<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
    <soapenv:Body/>
</soapenv:Envelope>
==============

==============
Listen Port: 9710
Target Host: GA2550
Target Port: 9700
==== Request ====
POST /orabpel/default/Customer/1.1/Supplier/SupplierRequester HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/#axisVersion#
Host: GA2550:9710
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: "orderChangeResponse"
Content-Length: 509
Connection: close

<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
    <soapenv:Header>
        <RelatesTo
```

```
            xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing"
        >
            bpel://localhost/default/Customer~1.1/301-BpInv0-BpSeq0.3-3
        </RelatesTo>
    </soapenv:Header>
    <soapenv:Body>
        <SupplierProcessResponse
            xmlns="http://services.qut.com/Supplier"
        >
            <nofItems>
                3
            </nofItems>
        </SupplierProcessResponse>
    </soapenv:Body>
</soapenv:Envelope>
==== Response ====
HTTP/1.1 200 OK
Date: Thu, 20 Oct 2005 11:58:43 GMT
Server: Oracle Application Server Containers for J2EE 10g (10.1.2.0.0)
Connection: Close
Content-Type: text/xml; charset=utf-8

<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
    <soapenv:Body/>
</soapenv:Envelope>
==============
```