Soundness Verification of Business Processes Specified in the Pi-Calculus

Frank Puhlmann

Business Process Technology Group
Hasso Plattner Institut for IT Systems Engineering
University of Potsdam
D-14482 Potsdam, Germany
frank.puhlmann@hpi.uni-potsdam.de

Abstract. Recent research in the area of business process management (BPM) introduced the application of a process algebra—the π-calculus—for the formal description of business processes and interactions among them. Especially in the area of service-oriented architectures, the key architecture for today's BPM systems, the π-calculus—as well as other process algebras—have shown their benefits in representing dynamic topologies. What is missing, however, are investigations regarding the correctness, i.e. soundness, of process algebraic formalizations of business processes. Due to the fact that most existing soundness properties are given for Petri nets, these cannot be applied. This paper closes the gap by giving characterizations of invariants on the behavior of business processes in terms of bisimulation equivalence. Since bisimulation equivalence is a well known concept in the world of process algebras, the characterizations can directly be applied to π-calculus formalizations of business processes. In particular, we investigate the characterization of five major soundness properties, i.e. easy, lazy, weak, relaxed, and classical soundness.

1 Introduction

Process algebras, which are algebraic frameworks for the study of concurrent processes, recently gained extended attention in the area of business process management (BPM), e.g. [1,2,3,4,5,6,7]. This is especially true within the area of service-oriented architecture (SOA) [8], which is today's standard architectural style for realizing BPM solutions [9,10]. What the existing approaches lack, however, is a distinguished investigation on correctness properties for the business processes they describe. By correctness properties, we refer to the different kinds of soundness that have been introduced in the workflow management domain and later on refined. In particular, these are the original soundness definition by van der Aalst [11], nowadays denoted as classical soundness, relaxed soundness by Dehnert [12], and weak soundness by Martens [13]. Noteworthy, all these properties cannot directly be applied to process algebraic formalizations of business processes, since they are characterized using Petri nets. Furthermore,

	Easy	Lazy	Weak	Relaxed	Classical
Possibility of termination	+	+	+	+	+
Support for lazy activities	+	+	-	+	-
Deadlock freedom	-	+	+	-	+
Participation of all activities	-	-	-	+	+

Table 1. Comparison of the different kinds of soundness.

liveness and boundedness are used to prove business processes formalized with Petri nets to be sound; both techniques which are not available for process algebraic verification.

The focus of our research is on the application of a special kind of process algebra—the π-calculus—to the domain of business process management [14,15]. This calculus is of special interest, since it supports a direct representation of dynamic binding as found in service-oriented architectures [16]. In this paper, however, we do not tackle dynamic binding but instead investigate the correctness of the "inner workings" of the different services found in a SOA. The behavior of the "inner workings" of a service is described as a business process composed out of common patterns [17]. During our research on the formal representation of these patterns, we developed a new soundness property, that proves a business process to be lazy sound if it always provides a result [18]. While lazy soundness provides one way of proving an invariant of a business process represented in a process algebraic formalization, there exists no investigation that discusses the verification of business processes according to existing soundness properties. In a practical setting, however, different properties might be required. A comparison of the different kinds of soundness is shown in table 1.

Since the availability of correctness properties is a fundamental constraint for any formalization of business processes [19], we close the gap for π-calculus mappings by providing means to characterize soundness using bisimulation equivalence. Stated simply, a bisimulation is an equivalence relation between two processes, where the actions of both processes are matched, i.e. if one process can do an action, there exists a matching action in the other process and vice versa. Beyond providing characterizations for existing soundness properties, we discuss the declaration and reasoning on arbitrary invariants for the behavior of business processes using bisimulation equivalence. The discussion provides the reader with the theoretical equipment to his or her tailored soundness property.

The paper is organized as follows. We start with a short introduction to the π-calculus in section 2, followed by a discussion of how invariants for the behavior of business processes can be represented and proved. Section 3 provides characterizations of five major soundness properties in the π-calculus. The practical applicability of bisimulation equivalence is shown in section 4. Finally, we conclude with a discussion of related work in section 5.

2 Preliminaries

We start with an introduction to the π-calculus and introduce how business processes can be formalized using this algebra. Thereafter we discuss the representation of weak invariants, i.e. properties that have to be fulfilled by some instances of a business process, and strong invariants, i.e. properties that have to be fulfilled by all instances.

2.1 The Pi-Calculus

The π-calculus is a process algebra for the formal description and analysis of concurrent, interacting processes, denoted as agents. The calculus is based on names, that represent the unification of channels and data, used by agents defined according to [20].

Definition 1 (Pi-Calculus). *The syntax of the π-calculus is given by:*

$$\begin{aligned} P &::= M \mid P|P \mid \nu z\ P \mid A(y_1, \ldots, y_n) \\ M &::= \mathbf{0} \mid \pi.P \mid M + M \\ \pi &::= \overline{x}\langle\tilde{y}\rangle \mid x(\tilde{z}) \mid \tau \mid [x = y]\pi\ . \end{aligned}$$

P and M denote the agents and summations of the calculus. The informal semantics is as follows: $P|P$ is the concurrent execution of P and P, $\nu z\ P$ is the restriction of the scope of the name z to P, i.e. z is only visible in P and distinct from all other names, and $A(y_1, \cdots, y_n)$ denotes parametric recursion over the set of agent identifiers. $\mathbf{0}$ is inaction, a process that can do nothing, and $M+M$ is the exclusive choice between M and M. The prefixes of the calculus are given by π. The output prefix $\overline{x}\langle\tilde{y}\rangle.P$ sends a tuple of names $\tilde{y}$ via the co-name $\overline{x}$ and then continues as P. The input prefix $x(\tilde{z})$ receives a tuple of names via the name x and then continues as P with $\tilde{z}$ replaced by the received names. Matching input and output prefixes of different components might communicate, leading to an intraaction that is unobservable (τ). An explicit representation of an unobservable action is given by the prefix $\tau.P$ and the match prefix $[x = y]\pi.P$ behaves as $\pi.P$ if x is equal to y. Throughout this paper, upper case letters are used for agent identifiers and lower case letters for names. We abbreviate a set of components as $\prod_{i=1}^{n} Pi$, i.e. $\prod_{i=1}^{3} Pi = P_1 \mid P_2 \mid P_3$.

2.2 Business Process Formalizations

Informally, a business process can be seen as a a special process that creates a value or a result for a customer. A business process is characterized by activities and control flow relations between them. Additional ingredients are the types of the activities, if they route the control flow, or additional attributes, mostly related to control flow decisions. For reasoning on soundness, we abstract from other perspectives. Formally, a business process can be represented by a process graph given by:

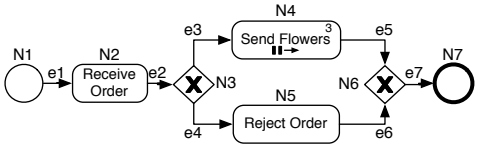

Fig. 1. A sample business process in BPMN notation.

Definition 2 (Process Graph). *A process graph is a four-tuple consisting of nodes, directed edges, types, and attributes. Formally:* $P = (N, E, T, A)$ *with*

- N *as a finite, non-empty set of nodes,*
- $E \subseteq (N \times N)$ *as a set of directed edges between nodes,*
- $T : N \rightarrow TYPE$ *as a function from nodes to types, and*
- $A \subseteq (N \times (KEY \times VALUE))$ *as a relation from nodes to key/value pairs.*□

N represents activities, incl. those responsible for routing the control flow given by E. T relates nodes with workflow patterns [21], where we assume simple activities or tasks to match the sequence pattern. A maps additional key/value pairs to nodes. Notable, a process graph only describes the static structure, i.e. the schema, of a business process. A process graph can easily be related to graphical notations such as EPCs [22], UML activity diagrams [23], or BPMN [24]. To give a process algebraic semantics to a process graph, the following, sketched algorithm is used (details can be found in [18]):

Algorithm 1 (Mapping Process Graphs to Agents). A process graph $P = (P_N, P_E, P_T, P_A)$ is mapped to π-calculus agents as follows:

1. All nodes of P are assigned a unique π-calculus agent identifier $N1 \ldots N|P_N|$.
2. All edges of P are assigned a unique π-calculus name $e1 \ldots e|P_E|$.
3. The π-calculus agents are defined according to the process patterns found in [17]. The functional perspective is represented by $\langle\cdot\rangle$. If the process graph is cyclic, recursion has to be used to allow multiple instances of activities.
4. An agent $N \stackrel{def}{=} (\nu e1, \ldots, e|P_E|\,)(\prod_{i=1}^{|P_N|} Ni)$ representing a process instance is defined. This agent might contain further components or restricted names according to the contained patterns. □

The given mapping of a process graph represents a single instance (case) of a business process. During the evolution of the agent terms, their structure is reduced to future states, whereas all (possible) past states are lost. We showcase the formalization of an example given in figure 1, where we provide a graphical representation of the process graph using BPMN. Contained is a simple business process of a flower shipper. The shipper receives an order and either accepts the order, and sends flowers to three participants given in the order, or the order is rejected. The flowers are sent asynchronously via a multiple instance task without synchronization (denoted by the arrow at the bottom of the activity).

The formalization of the business process starts by assigning unique agent identifiers and names to the nodes and edges. These are already shown in the figure. The corresponding agents are given according to step three of algorithm 1 as follows:

$$N1 \stackrel{def}{=} \langle\cdot\rangle.\overline{e1}.\mathbf{0} \text{ and } N7 \stackrel{def}{=} e7.\langle\cdot\rangle.\mathbf{0}$$

represent the start and the end event. The functional perspective is abstracted by $\langle\cdot\rangle$, which will be filled later on. The nodes of the type task are given by the sequence workflow pattern:

$$N2 \stackrel{def}{=} e1.\langle\cdot\rangle.\overline{e2}.\mathbf{0} \text{ and } N5 \stackrel{def}{=} e4.\langle\cdot\rangle.\overline{e6}.\mathbf{0} \ .$$

The exclusive split and join gateways are given by

$$N3 \stackrel{def}{=} e2.\langle\cdot\rangle.(\overline{e3}.\mathbf{0} + \overline{e4}.\mathbf{0}) \text{ and } N6 \stackrel{def}{=} e5.\langle\cdot\rangle.\overline{e7}.\mathbf{0} + e6.\langle\cdot\rangle.\overline{e7}.\mathbf{0} \ .$$

The multiple instance task with three static instances is given by

$$N4 \stackrel{def}{=} e3.(\langle\cdot\rangle.\mathbf{0} \mid \langle\cdot\rangle.\mathbf{0} \mid \langle\cdot\rangle.\mathbf{0} \mid \overline{e5}.\mathbf{0}) \ .$$

Finally, an agent

$$N \stackrel{def}{=} (\nu e1, \ldots, e7\,)(\prod_{i=1}^{7} Ni) \ .$$

represents a fresh process instance. An extended discussion of the semantics and the mapping can be found in [17,18].

2.3 Simulation

To specify that a process instance given by π-calculus agents *can* fulfill an invariant, we need the concept of simulation from the process algebraic toolbox. Informally, a simulation relates an agent P with another agent Q, if Q can follow every action of P. We say that Q can simulate P. The actions of the agents are given by the input, output, and unobservable prefixes, denoted as $Act = \{\overline{x}\langle\tilde{y}\rangle, x(\tilde{z}), \tau\}$. The evolution of the state of an agent to a succeeding state is denoted by a transition bearing the corresponding action, i.e. $\overline{a}\langle w\rangle.A \xrightarrow{\overline{a}\langle w\rangle} A$.

Definition 3 (Simulation). *A simulation is a binary relation $\mathcal{R}$ on agents such that $\forall\alpha \in Act$:*

$$P\mathcal{R}Q \wedge P \xrightarrow{\alpha} P' \Rightarrow \exists Q' : Q \xrightarrow{\alpha} Q' \wedge P'\mathcal{R}Q' \ .$$

Q is similar to P, denoted as $P \precsim Q$, if they are related by a simulation.

Simulation considers a strong relation between interactions and unobservable actions. Two agents

$$P \stackrel{def}{=} a(x).\tau.\tau.\overline{b}\langle z\rangle.\mathbf{0} \text{ and } Q \stackrel{def}{=} a(x).\tau.\overline{b}\langle z\rangle.\mathbf{0}$$

cannot simulate each other, since they differ in the number of their unobservable actions (τ transitions). A simulation that abstracts from these unobservable actions is called weak simulation. Weak simulations are of particular interest, since they abstract from the internal behavior of agents and instead only consider the external visible behavior. A weak simulation is obtained by defining $\Longrightarrow$ to represent zero or more τ transitions, i.e. $\stackrel{\tau}{\longrightarrow}^*$, $\stackrel{\alpha}{\Longrightarrow}$ as $\Longrightarrow\stackrel{\alpha}{\longrightarrow}\Longrightarrow$, and $\stackrel{\hat{\alpha}}{\Longrightarrow}$ as $\stackrel{\alpha}{\Longrightarrow}$ if $\alpha \neq \tau$ and $\Longrightarrow$ if $\alpha = \tau$.

Definition 4 (Weak Simulation). *A weak simulation is a binary relation $\mathcal{R}$ on agents such that $\forall \alpha \in Act$:*

$$P \mathcal{R} Q \wedge P \stackrel{\alpha}{\longrightarrow} P' \Rightarrow \exists Q' : Q \stackrel{\hat{\alpha}}{\Longrightarrow} Q' \wedge P' \mathcal{R} Q' \ .$$

Q is weak similar to P, denoted as $P \precsim Q$, if they are related by a weak simulation.

By using weak simulation, we can match π-calculus mappings of process graphs for the potential fulfillment of arbitrary invariants. Therefore we specify the invariant by a π-calculus agent that trivially fulfills them. Consider for instance

$$I1 \stackrel{def}{=} \overline{s}.\mathbf{0} \ ,$$

that denotes that an activity can occur during the execution of a process instance via an emission of $\overline{s}$. To enhance the agent that represents the activity under investigation in the π-calculus mapping, we need to replace $\langle \cdot \rangle$ by $\overline{s}$. Let's assume we want to prove that the reject order task from figure 1 can be executed. Hence, we change agent $N5$ accordingly:

$$N5 \stackrel{def}{=} e4.\overline{s}.\overline{e6}.\mathbf{0} \ .$$

We assume all other functional abstractions to be filled with unobservable actions (τ). Now we can decide if $I1 \precsim N$ holds by finding a relation $\mathcal{R}$ that relates both agents. If we are able to find such a relation, we proved that the reject order task can indeed be executed. If we found a counterexample, i.e. something that *I1* can do but N is unable to simulate, we disproved the invariant for the business process. The magic of weak simulation lies in the fact that all internal interactions between components, i.e. between *N1* and *N2* via *e1* are unobservable from the outside, meaning they resemble τ. Since all names $e*$ are restricted inside N, only $\overline{s}$ has to be considered in the simulation. Regarding this semantics, the agent N breaks down to a number of τ actions, a possible emission via $\overline{s}$, and another number of τ actions. Hence, it resembles $\stackrel{\hat{\overline{s}}}{\Longrightarrow}$. According to definition 4, $\mathcal{R}$ is given by $\{(I1, N), (\mathbf{0}, \mathbf{0})\}$ and $I1 \precsim N$ holds. Thus, the business process from figure 1 fulfills the invariant that the reject order task can be executed.

We can also disprove invariants for business processes, i.e. show that the reject order activity will never be executed twice per instance. Therefore we modify the agent representing the invariant to

$$I2 \stackrel{def}{=} \overline{s}.\overline{s}.\mathbf{0} \ .$$

Due to the fact that $I2 \xrightarrow{\overline{s}} \overline{s}.\mathbf{0}$, and correspondingly $N \stackrel{\hat{\overline{s}}}{\Longrightarrow} \mathbf{0}$, the remainder of *I2* can do another action $\overline{s}$ that the remainder $\mathbf{0}$ of N cannot simulate. Hence, $I2 \not\precapprox N$ disproves the supposed invariant.

2.4 Bisimulation

By using weak simulation we are able to show an optional behavior. This is due to the fact that a simulation only investigates one direction. If the agent mapping of a process graph contains additional actions, it is not investigated if the agent representing the invariant can mimic them. To enforce that two agents are able to mimic all their actions in arbitrary directions, i.e. the first agent does something, the second agent corresponds, and thereafter the second agent does something else that the first agent needs to mimic, etc., we require the relation $\mathcal{R}$ to be symmetric. Helpful is again the weak variant, yielding weak bisimulation:

Definition 5 (Weak Bisimulation). *A weak bisimulation is a symmetric, binary relation $\mathcal{R}$ on agents such that $\forall \alpha \in Act$:*

$$P \mathcal{R} Q \wedge P \xrightarrow{\alpha} P' \Rightarrow \exists Q' : Q \stackrel{\hat{\alpha}}{\Longrightarrow} \alpha Q' \wedge P' \mathcal{R} Q' \ .$$

P and Q are weak bisimilar, denoted as $P \approx Q$, if they are related by a weak bisimulation.

According to this definition, weak bisimulation equivalence, or weak bisimilarity, between two agents P and Q is stronger than mutual simulation, i.e. from $P \approx Q \Rightarrow P \precapprox Q \wedge Q \precapprox P$, but the converse does not necessarily hold. Consider for instance once again *I1* and N as given in the previous section. While $I1 \precapprox N$ holds (as shown), also $N \precapprox I1$ holds (proof left for the reader). However, this does not mean that the reject order task is executed in every instance, as can be easily checked in figure 1. The technical difference lies in the fact that bisimulation is symmetric, a property that allows switching the direction after every step instead assuming a fixed order. According to bisimulation, we need to find a counterexample to prove $I1 \not\approx N$ and thereby disprove the proposition that reject order is executed in every instance. Since this is a quite complex task, we refer the reader to section 4, where we introduce tool-supported reasoning (indeed, a counterexample can be found).

The last application of bisimulation that we will consider is proving invariants that hold for all instances. Regarding figure 1, it seems obvious that receive order is executed in every instance. We can prove this proposition by returning to the original definition of N and modify its component *N2*, representing the receive order task, accordingly:

$$N2 \stackrel{def}{=} e1.\overline{s}.\overline{e2}.\mathbf{0} \ .$$

By finding a relation $\mathcal{R}$ between *I1* and the modified N according to definition 5 we can prove that receive order is executed in each instance. Since such an relation exists, $I1 \approx N$ holds (for the modified N). As the relation contains 29 tuples, we once again refer to section 4 for tool-supported reasoning.

3 Characterizations of Soundness

After having shown how *can* properties of business processes are proved using simulation and *must* properties are proved using bisimulation, we discuss existing soundness properties. Since most existing properties are given for workflow nets [25], a subclass of Petri nets, we define a subset of process graphs that fulfills the same structural properties. We denote this property as *structural soundness*. Informally, structural soundness is given by:

> A process graph is structural sound if it has exactly one initial node, exactly one final node, and all other nodes lie on a path between the initial and the final node.

Structural sound process graphs resemble placeholders for business processes with the denoted structural properties. We omit the (obvious) formal definition due to space limits.

3.1 Easy Soundness

The least soundness property a business process should fulfill is given by *easy soundness*, informally given by:

> A structural sound process graph representing a business process is easy sound if a result can be provided.

As indicated by the word *can*, we have to use simulation to prove this property for process algebraic formalizations of business processes. In particular, we have to be able to observe the occurrence of the initial and the final node. The idea is depicted in figure 2. A structural sound process graph is fed into a black box. Each time we press the start button, an instance of the process graph is executed. Each time the final node of process graph is executed, the done bulb flashes. Regarding easy soundness, we have to find at least one process instance where the done bulb flashes, denoting the delivery of the result. An agent fulfilling this invariant is given by:

$$S_{EASY} \stackrel{def}{=} i.\tau.\overline{o}.\mathbf{0} \; . \tag{1}$$

The input prefix i denotes the pushbutton, whereas the output prefix $\overline{o}$ resembles the done bulb. Both are in a fixed sequence, i.e. $\overline{o}$ follows always after i. The τ in-between denotes the abstraction from complex actions. Since we use weak (bi)-simulations, however, it could also be omitted. To be able to decide whether a business process given by π-calculus agents is weak similar to S_{EASY}, we have to enhance the agents representing the business process:

Algorithm 2 (Easy Soundness Annotated Pi-Calculus Mapping). The π-calculus mapping of a process graph according to algorithm 1 is enhanced for reasoning on easy soundness as follows. The functional abstraction $\langle\cdot\rangle$ of (1) the agent that represents the initial node is replaced by $i.\tau$; (2) the agent that represents the final node is replaced by $\tau.\overline{o}$; (3) all other agents are replaced by

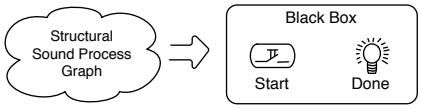

Fig. 2. Black box investigation of a structural sound process graph.

τ. Obviously, i and o are not permitted to appear anywhere else in the agent terms. □

A formal definition of easy soundness using weak similarity is now given by:

Definition 6 (Easy Sound Process Graph). *A structural sound process graph P with a semantics given by the easy soundness annotated π-calculus mapping D of P is easy sound if $S_{EASY} \precsim D$ holds.*

We can prove the sample business process from figure 1 to be easy sound by finding a relation for $S_{EASY} \precsim N_{EASY}$, with N_{EASY} being syntactically equal to N with $\langle\cdot\rangle$ replaced by τ and:

$$N1 \stackrel{def}{=} i.\tau.\overline{e1}.\mathbf{0} \text{ and } N7 \stackrel{def}{=} e7.\tau.\bar{o}.\mathbf{0}\ .$$

Since such a relation exists (4 tuples), the business process of the flower shipper is easy sound. The relation can be reconstructed by the reader as will be shown in section 4.

3.2 Lazy Soundness

One obvious extension to easy soundness is given by enforcing that all instances of a process graph provide a result:

> A structural sound process graph representing a business process is lazy sound if in any case a result is provided exactly once.

This property can be proved using weak bisimilarity. Furthermore, all assumptions from easy soundness also hold. In particular, we need to be able to observe the occurrence of the final node after *each* occurrence of the initial node. Regarding the black box from figure 2, this means that each time the start button is pressed, we need to be able to observe exactly one flash of the done bulb. In contrast to easy soundness, we cannot try until we observe a flash of the done bulb, but have to consider all possibilities instead. This supplies two problems: (1) How can we be sure that all path of the process graph have been traversed, i.e. we do not need to press the start button anymore; (2) How do we know if we do not need to wait any longer for the done bulb to flash, i.e. a deadlock has occurred? If we are able to find a bisimulation between an invariant given by

$$S_{LAZY} \stackrel{def}{=} i.\tau.\bar{o}.\mathbf{0} \tag{2}$$

and an annotated agent mapping of a process graph, both problems have been overcome. The former due to the fact that a bisimulation enumerates all possible states and the latter by the fact that a bisimulation is finite. Since S_{LAZY} exactly resembles S_{EASY}, the same annotation for the agents has to be used:

Algorithm 3 (Lazy Soundness Annotated Pi-Calculus Mapping). The same as given by algorithm 2. □

A formal definition of lazy soundness using weak bisimilarity is now given by:

Definition 7 (Lazy Sound Process Graph). *A structural sound process graph P with a semantics given by the lazy soundness annotated π-calculus mapping D of P is lazy sound if $D \approx S_{LAZY}$ holds.*

The business process from figure 1 can be checked for satisfying lazy soundness; i.e. we need to prove that $S_{LAZY} \approx N_{EASY}$ holds.

3.3 Weak Soundness

Lazy soundness only considers the return of a result, whereas the termination of the process graph, i.e. all nodes are terminated, is not considered. This is due to the fact that deferred, so called *lazy*, activities can remain active after the result has been provided (an extended discussion can be found in [18]). Nevertheless, in some cases it has to be guaranteed that the termination of a business process occurs the very moment the result is provided:

> A structural sound process graph representing a business process is weak sound if in any case a result is provided and the process instance is terminated the moment the result is provided.

Since once again all cases need to be considered, weak bisimilarity is the technique of choice. What has to be changed, however, is the black box we use for investigation. In addition to be able to observe the occurrence of the initial and the final node, we also need to be able to observe the occurrence of each other node. This enhancement is depicted in figure 3. A business process placed inside the enhanced black box is weak sound if we are unable to observe a flash of the step bulb after a flash of the done bulb. Furthermore, a flash of the done bulb has to be observed exactly once for each push on the start button. Deriving the invariant is not this easy, however. A naive version given by

$$I \stackrel{def}{=} i.I1 \quad \text{and} \quad I1 \stackrel{def}{=} \overline{s}.I1 + \tau.\overline{o}.\mathbf{0}$$

will not work with an annotated π-calculus mapping given below (the proof is left to the reader; hint: *I1* can send an unlimited times via $\overline{s}$). The problem can be overcome by allowing each instance of a process graph to emit via $\overline{s}$ only once. The choice which node will emit via $\overline{s}$ has to be made *non-deterministically*. This is done via an activity observation agent that will be included in the π-calculus mapping of a process graph later on:

$$\begin{aligned} X(x,s) &\stackrel{def}{=} x(ack).(\tau.\overline{ack}.\mathbf{0} \mid X(x,s)) + x(ack).(\tau.\overline{s}.\overline{ack}.\mathbf{0} \mid X_1(x)) \\ X_1(x) &\stackrel{def}{=} x(ack).(\tau.\overline{ack}.\mathbf{0} \mid X_1(x)) \ . \end{aligned} \tag{3}$$

Agent X is triggered via x and thereafter has the non-deterministic choice between acknowledging via ack or emitting via $\overline{s}$ and thereafter acknowledging. If the former happened, X behaves again as X. If the latter happened, X behaves as $X1$ that is only capable of acknowledging. Due the activity observation agent, each node of a process graph has the capability of signaling its execution. As this explicitly includes nodes of a process graph that are active after the final node has signaled its execution, weak soundness can be proved by an invariant given as

$$S_{WEAK} \stackrel{def}{=} i.(\tau.\overline{o}.\mathbf{0} + \tau.\overline{s}.\overline{o}.\mathbf{0}) \; . \tag{4}$$

S_{WEAK} is the same as S_{LAZY} regarding i and $\overline{o}$. After the observation of the initial node via i, a choice between observing $\overline{o}$ or $\overline{s}$ is made. If $\overline{o}$ is observed, no other observations are possible (due to S_{WEAK} becomes inaction). If $\overline{s}$ is observed, the next observation has to be $\overline{o}$. Thereafter, no other observations are possible. This behavior resembles the enhanced black box with the exception that the step bulb might flash only once before/after the done bulb flashed. The agents that represent the business process have to be enhanced as given by the following algorithm:

Algorithm 4 (Weak Soundness Annotated Pi-Calculus Mapping). The π-calculus mapping D of a process graph $P = (N, E, T, A)$ according to algorithm 1 is enhanced for reasoning on weak soundness as follows. The functional abstraction $\langle\cdot\rangle$ of (1) the agent that represents the initial node is replaced by $\nu ack\; i.\overline{x}\langle ack\rangle.ack$; (2) the agent that represents the final node is replaced by $\nu ack\; \overline{x}\langle ack\rangle.ack.\overline{o}.\tau$; (3) all other agents are replaced by $\nu ack\; \overline{x}\langle ack\rangle.ack$. Furthermore, the agent from equation 3 has to be included in D:

$$D \stackrel{def}{=} (\nu e1, \ldots, e|E|, x)(\prod_{i=1}^{|N|}(Di) \mid X) \; .$$

The names i, o, and s are not permitted to appear anywhere else in the agent terms. □

A formal definition of weak soundness for a process graph is now given by:

Definition 8 (Weak Sound Process Graph). *A structural sound process graph P with a semantics given by the weak soundness annotated π-calculus mapping D of P is weak sound if $D \approx S_{WEAK}$ holds.*

The business process from figure 1 is not fulfilling weak soundness. This is due to the fact that the flowers are sent asynchronously; i.e. the send flowers activity is lazy.

3.4 Relaxed Soundness

All preceding soundness properties neglect an investigation regarding the participation of activities in a business process. Sometimes this is an important property, due to the fact that unused activities can be removed from a business process. Similar to [12], our interpretation of *relaxed soundness* also supports the synchronizing merge pattern:

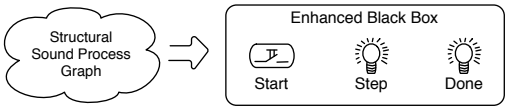

Fig. 3. Enhanced black box investigation of a structural sound process graph.

> A structural sound process graph representing a business process is relaxed sound if each node of the process graph has the possibility of being executed in between the execution of the initial and the final node.

Since we talk about a possibility, weak similarity has to be used this time. We can reuse the enhanced black box from figure 3 with a special preparation of the π-calculus mapping. In particular, we need to prepare as much copies of the π-calculus mapping as there are nodes in the process graph. In each copy we need to give another node the possibility of emitting via $\overline{s}$ to signal its execution. Hence, for each enhanced mapping of a process graph feed into the enhanced black box, we should be able to observe a flash of the step and done bulb in sequence in at least one pass. The invariant is given by:

$$S_{RELAXED} \stackrel{def}{=} i.\overline{s}.\overline{o}.\mathbf{0} \ .$$

The agent $S_{RELAXED}$ simply resembles the execution order. However, special care has to be taken if the node under observation is always executed more than once (the next action of $S_{RELAXED}$ after $\overline{s}$, $\overline{o}$, cannot be simulated by a mapping that emits $\overline{s}$ once again!). This problem can be solved by including an activity loop observation agent that only emits $\overline{s}$ for the first execution of a node:

$$Y(y,s) \stackrel{def}{=} y(ack).\overline{s}.\overline{ack}.Y_1(y) \ \text{ and } \ Y_1(y) \stackrel{def}{=} y(ack).\tau.\overline{ack}.Y_1(y) \ .$$

The agent is integrated in the mapping of a process graph as follows:

Algorithm 5 (Relaxed Soundness Annotated Pi-Calculus Mapping). To annotate a π-calculus mapping D of a process graph $P = (N, E, T, A)$ for a certain node $n \in N \setminus \{x, y\}$ with $T(x) = \mathit{InitialNode}, T(y) = \mathit{FinalNode}$ for reasoning on relaxed soundness regarding the node n, the following steps have to be made. The functional abstraction $\langle\cdot\rangle$ of (1) the agent that represents the initial node is replaced by $i.\tau$; (2) the agent that represents the final node is replaced by $\tau.\overline{o}$; (3) the agent that represents n is replaced by $\nu ack\ \overline{y}\langle ack\rangle.ack$; (4) all other agent is replaced by τ. Furthermore, the agent from equation 3.4 has to be included in D:

$$D \stackrel{def}{=} (\nu e1, \ldots, e|E|, y)(\prod_{i=1}^{|N|}(Di) \mid Y) \ .$$

The names i, o, and s are not permitted to appear anywhere else in the agent terms. □

Relaxed soundness is formally given by:

Definition 9 (Relaxed Sound Process Graph). *A structural sound process graph $P = (N, E, T, A)$ is relaxed sound if for each relaxed soundness annotated π-calculus mapping D considering $n \in N \backslash \{x, y\}$ with $T(x) = InitialNode, T(y) = FinalNode$ it holds that $S_{RELAXED} \precapprox D$.*

Regarding the example from figure 1, it can be shown that each node has the possibility to participate in the business process by calculating the corresponding weak simulations.

3.5 Classical Soundness

A strong soundness property, known as (classical) soundness [11], is given informally by:

> A structural sound process graph representing a business process is sound if (1) in any case a result is provided; (2) the process instance is terminated the moment the result is provided; and (3) each node of the process graph has the possibility of being executed after the initial node.

The first two criteria coincidence with weak soundness. The last criterion is given by a modified version of relaxed soundness, where the activity loop observation agent and $\overline{o}$ are omitted from the π-calculus mapping. The relaxed invariant is given by:

$$S_{PART} \stackrel{def}{=} i.\overline{s}.\mathbf{0} \tag{5}$$

We can omit the activity loop observation agent due to the fact that multiple emissions via $\overline{s}$ from the π-calculus mapping of the process graph are not disturbing the similarity because $\overline{o}$ is omitted.

Algorithm 6 (Participating Annotated Pi-Calculus Mapping). To annotate a π-calculus mapping of a process graph $P = (N, E, T, A)$ for a certain node $n \in N \setminus \{x, y\}$ with $T(x) = InitialNode, T(y) = FinalNode$ for reasoning on the participation of n in the business process, the following steps have to be made. The functional abstraction $\langle \cdot \rangle$ of (1) the agent that represents the initial node is replaced by $i.\tau$; (2) the agent that represents the node n is replaced by $\overline{s}$; (3) all other agents is replaced by τ. The names i and s are not permitted to appear anywhere else in the agent terms. □

The definition of classical soundness using weak similarity and bisimilarity is formally given by:

Definition 10 (Classical Sound Process Graph). *A structural sound process graph $P = (N, E, T, A)$ with a semantics given by (1) the weak soundness annotated π-calculus mapping D1 of P and (2) a set of participating annotated π-calculus mappings D2 for each $n \in N \setminus \{x, y\}$ with $T(x) = InitialNode, T(y) = FinalNode$ is classical sound if it holds that (a) $D1 \approx S_{WEAK}$ and (b) $S_{PART} \precapprox D$ for each $D \in D2$.*

4 Tool Support and Efforts

This first part of this section shows the practical applicability of the different soundness characterization using an existing tool. The second part discusses an important criterion: The efforts required for deciding bisimulation equivalence for the different kinds of soundness.

4.1 Tool-supported Reasoning

Reasoning on similarity and bisimilarity of π-calculus agents can be done using the Advanced Bisimulation Checker (ABC).[1] The reasoner accepts agents in an ASCII syntax described in the corresponding documentation. In a nutshell, `'x<y>` represents an output prefix, `x(y)` an input prefix, `t` an unobservable action, and `(^z)` the restriction operator. Regarding the easy/lazy soundness annotated mapping from figure 1, the following input is appropriate:

```
agent N1(e1,i)=i.t.'e1.0
agent N2(e1,e2)=e1.t.'e2.0
agent N3(e2,e3,e4)=e2.t.('e3.0 + 'e4.0)
agent N4(e3,e5)=e3.(t.0 | t.0 | t.0 | 'e5.0)
agent N5(e4,e6)=e4.t.'e6.0
agent N6(e5,e6,e7)=e5.t.'e7.0 + e6.t.'e7.0
agent N7(e7,o)=e7.t.'o.0
agent N(i,o)=(^e1,e2,e3,e4,e5,e6,e7)(N1(e1,i) | N2(e1,e2) | N3(e2,e3,e4) | N4(e3,e5) |
      N5(e4,e6) | N6(e5,e6,e7) | N7(e7,o))
agent S_EASY(i,o)=i.t.'o.0
agent S_LAZY(i,o)=i.t.'o.0
```

Easy soundness can be decided by asking ABC for proving similarity between S_{EASY} and N using the `wlt` command:

```
abc > wlt S_EASY(i,o) N(i,o)
The two agents are weakly related (4).
```

Since a simulation exists, easy soundness for the business process from figure 1 has been proved. Lazy soundness can be decided by proving bisimilarity between S_{LAZY} and N using the `weq` command:

```
abc > weq S_LAZY(i,o) N(i,o)
The two agents are weakly related (70).
```

Both agents are bisimilar due to the fact that a bisimulation has been found. A session disproving weak soundness for the example is given by:

```
agent N1(e1,i,x)=i.(^ack)'x<ack>.ack.'e1.0
agent N2(e1,e2,x)=e1.(^ack)'x<ack>.ack.'e2.0
agent N3(e2,e3,e4,x)=e2.(^ack)'x<ack>.ack.('e3.0 + 'e4.0)
agent N4(e3,e5,x)=e3.((^ack)'x<ack>.ack.0 | (^ack)'x<ack>.ack.0 | (^ack)'x<ack>.ack.0 | 'e5.0)
agent N5(e4,e6,x)=e4.(^ack)'x<ack>.ack.'e6.0
agent N6(e5,e6,e7,x)=e5.(^ack)'x<ack>.ack.'e7.0 + e6.(^ack)'x<ack>.ack.'e7.0
agent N7(e7,o,x)=e7.(^ack)'x<ack>.ack.'o.0
agent X(x,s)=x(ack).(t.'ack.0 | X(x,s)) + x(ack).('s.'ack.0 | X_1(x))
agent X_1(x)=x(ack).(t.'ack.0 | X_1(x))
```

[1] Available at `http://lampwww.epfl.ch/~sbriais/abc/abc.html`.

```
agent N(i,o,s)=(^e1,e2,e3,e4,e5,e6,e7,x)(N1(e1,i,x) | N2(e1,e2,x) | N3(e2,e3,e4,x) |
      N4(e3,e5,x) | N5(e4,e6,x) | N6(e5,e6,e7,x) | N7(e7,o,x) | X(x,s))
agent S_WEAK(i,o,s)=i.(t.'o.0 + t.'s.'o.0)

abc > weq S_WEAK(i,o,s) N(i,o,s)
The two agents are not weakly related (30).
```

Further examples are omitted due to a lack of space.

4.2 Efforts

This subsection takes a closer look at the complexity of deciding bisimulation. In the general case, bisimulation equivalence on π-calculus agents is undecidable. This is due to the Turing-completeness of the calculus, e.g. shown in [20]. What can be decided, however, is non-equivalence of agents, since after finite number of transitions, a counterexample has to be found. Nevertheless, our aim is to prove that a π-calculus mapping of a business process fulfills a certain property, hence it is equivalent.

The problems can partly be overcome by restricting the grammar of the π-calculus variant applied. For the following discussion, we consider a business process with a number of nodes, given by the following agent:

$$N \stackrel{def}{=} (e1, e2, \dots)(\prod_{i=1} Ni) \ .$$

Agents with Simple Sequences. Simple sequences, such as

$$N1 \stackrel{def}{=} \langle\cdot\rangle.\overline{e1}.\mathbf{0} \ , \ \ N2 \stackrel{def}{=} e1.\langle\cdot\rangle.\overline{e2}.\mathbf{0} \ \text{ and } \ N3 \stackrel{def}{=} n2.\langle\cdot\rangle.\mathbf{0} \ ,$$

can be enforced by removing recursion via defined agent identifiers from the calculus. As a result, loops are prohibited. This significantly drops the effort for most practical problem sizes. However, we also loose Turing-completeness.

Agents Mappings with Loops in the Business Processes. Agents that represent business processes with loops, such as

$$N1 \stackrel{def}{=} \langle\cdot\rangle.\overline{e1}.\mathbf{0} \ , \ \ N2 \stackrel{def}{=} e1.\langle\cdot\rangle.((\overline{e1}.\mathbf{0} + \overline{e2}.\mathbf{0}) \mid N2) \ \text{ and } \ N3 \stackrel{def}{=} n2.\langle\cdot\rangle.\mathbf{0} \ ,$$

can in most cases efficiently be checked, because the same state(s) appears over and over again. However, we do not allow the creation of restricted names in recursive passages, since this would lead to the next problem class.

Arbitrary Recursion and Restrictions. Agents such as

$$\begin{aligned} A &\stackrel{def}{=} a.(A_1(b) \mid A_2) \\ A_1(prev) &\stackrel{def}{=} \nu next\ \overline{create\_i}\langle next, prev\rangle.A_1(next) + \overline{prev}.\mathbf{0} \\ A_2 &\stackrel{def}{=} create\_i(next, prev).(\langle\cdot\rangle.next.\overline{prev}.\mathbf{0} \mid A_2) \ . \end{aligned}$$

where arbitrary restricted names can be created in recursive passages are hard to verify, because new states are created all the way. However, this kind of problem is only to be found in the multiple instances workflow patterns (as shown), which can be abstracted by τ for verification.

Agents with massive non-determinism. Agents such as X and Y according to weak and relaxed soundness contain massive amounts of non-determinism. This has an exponential influence on the state space that needs to be checked. Consequently, the most promising property regarding computational complexity is lazy soundness.

Solutions. Our current efforts go into the direction of implementing a domain-specific bisimulation checker for BPM. The already restricted input set given by process graphs is further stripped down by applying the asynchronous π-calculus [20], which is also able to represent all workflow patterns. The goal of our research is not limiting the input further, e.g. by only allowing block structures or prohibiting loops. Instead, we are working on a simplification of the workflow pattern formalization, the normalization and optimization of the generated agent term, as well as including heuristics via external data (e.g. process graphs). In this paper, we laid the formal foundations behind bisimulation-based soundness verification.

5 Conclusion and Related Work

In this paper we have shown how invariants for π-calculus mappings of business processes can be declared and proved. Besides introducing the general concepts in section 2, we also investigated easy, lazy, weak, relaxed, and classical soundness in section 3. The practical feasibility of our findings has been sketched afterwards in section 4, where we sketched the question of computational complexity. However, future research in this area is crucial for the practical applicability. In particular, we will investigate different classes of inputs vs. different soundness properties. While weak, relaxed, and classical soundness rely on link passing mobility, that is not available in all process algebras, the general concepts can also be applied to other algebras like CCS [26]. We already presented a tool chain for lazy soundness as part of earlier research [18,27]. This paper goes one step further by discussing the general concepts as well as missing soundness properties. To the knowledge of the authors, no other approach using similarity and bisimilarity for deciding different kinds of soundness has been published. Nevertheless, as we already sketched in [27], also projection inheritance [28] for Petri nets can be used.

Regarding foundational work, the different soundness definitions from van der Aalst [11], Dehnert [12], and Martens [13] directly inspired our definitions. Since these are given for Petri nets, we could only informally resemble them. For instance, the black box verification of lazy soundness closely resembles the first criterion of soundness for workflow nets:

$$\forall_M (i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o) \ .$$

It states that a workflow net has the option to always complete, i.e. deliver a result from our perspective. The second criterion,

$$\forall_M (i \xrightarrow{*} M \wedge M \geq o) \Rightarrow (M = o) \ ,$$

is resembled by weak soundness and the enhanced black box observation. It states that a workflow net terminates the moment a token is in the final place, i.e. the result is provided the moment the process instance is terminated. The third criterion,

$$\forall_{t \in T} \exists_{M,M'} i \xrightarrow{*} M \xrightarrow{t} M' \ ,$$

states that each task of a workflow net can participate in the workflow. It is resembled by a subset of relaxed soundness as described in section 3.5.

Remarks. The definition of weak bisimulation has been simplified, since otherwise more elaborate foundations for the π-calculus would be required (e.g. bound and free names). The reader is refered to [14].

References

1. Brogi, A., Canal, C., E.Pimentel, Vallecillo, A.: Formalizing Web Service Choreographies. In: Proceedings of First International Workshop on Web Services and Formal Methods. Electronic Notes in Theoretical Computer Science, Elsevier (2004)
2. Laneve, C., Zavattaro, G.: Foundations of Web Transactions. In Sassone, V., ed.: Foundations of Software Science and Computational Structures, volume 3441 of LNCS, Berlin, Springer Verlag (2005) 282–298
3. Bordeaux, L., Salaün, G.: Using Process Algebra for Web Services: Early Results and Perspectives. In Shan, M., Dayal, U., Hsu, M., eds.: Technologies for E-Services, volume 3324 of LNCS, Berlin, Springer Verlag (2005) 54–68
4. Guidi, C., Lucchi, R., Gorrieri, R., Busi, N., Zavattaro, G.: SOCK: A Calculus for Service Oriented Computing. In Dam, A., Lamersdorf, W., eds.: Service-Oriented Computing – ICSOC 2006, volume 4294 of LNCS, Berlin, Springer Verlag (2006) 327–338
5. Mazzara, M., Lanese, I.: Towards a Unifying Theory for Web Service Composition. In Bravetti, M., Núñez, M., Zavattaro, G., eds.: Web Services and Formal Methods, volume 4184 of LNCS, Berlin, Springer Verlag (2006) 257–272
6. Ferrara, A.: Web Services: A Process Algebra Approach. In: ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing, New York, NY, USA, ACM Press (2004) 242–251
7. Decker, G., Zaha, J., Dumas, M.: Execution Semantics for Service Choreographies. In Bravetti, M., Núñez, M., Zavattaro, G., eds.: Web Services and Formal Methods, volume 4184 of LNCS, Berlin, Springer Verlag (2006) 163–177
8. Burbeck, S.: The Tao of E-Business Services (2000)
9. Woodley, T., Gagnon, S.: BPM and SOA: Synergies and Challenges. In Ngu, A., Kitsuregawa, M., Neuhold, E., Chung, J., Sheng, Q., eds.: Web Information Systems Engineering – WISE 2005: 6th International Conference on Web Information Systems Engineering, volume 3806 of LNCS, Berlin, Springer Verlag (2005) 679–688

10. Newcomer, E., Lomov, G.: Understanding SOA with Web Services. Addison–Wesley (2005)
11. Aalst, W.: Verification of Workflow Nets. In Azéma, P., Balbo, G., eds.: Application and Theory of Petri Nets 1997, volume 1248 of LNCS, Berlin, Springer Verlag (1997) 407–426
12. Dehnert, J., Rittgen, P.: Relaxed Soundness of Business Processes. In Dittrich, K., Geppert, A., Norrie, M., eds.: anced Information Systems Engineering: 13th International Conference (CAiSE 2001), volume 2068 of LNCS, Berlin, Springer Verlag (2001) 157–170
13. Martens, A.: Analyzing Web Service based Business Processes. In Cerioli, M., ed.: Fundamental Approaches to Software Engineering (FASE'05), volume 3442 of LNCS, Springer Verlag (2005) 19–33
14. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, Part I/II. Information and Computation **100** (1992) 1–77
15. Puhlmann, F.: Why do we actually need the Pi-Calculus for Business Process Management? In Abramowicz, W., Mayr, H., eds.: 9th International Conference on Business Information Systems (BIS 2006), volume P-85 of LNI, Bonn, Gesellschaft für Informatik (2006) 77–89
16. Overdick, H., Puhlmann, F., Weske, M.: Towards a Formal Model for Agile Service Discovery and Integration. In: Proceedings of the International Workshop on Dynamic Web Processes (DWP 2005). IBM technical report RC23822, Amsterdam (2005)
17. Puhlmann, F., Weske, M.: Using the Pi-Calculus for Formalizing Workflow Patterns. In: Business Process Management, volume 3649 of LNCS, Berlin, Springer Verlag (2005) 153–168
18. Puhlmann, F., Weske, M.: Investigations on Soundness Regarding Lazy Activities. In Dustdar, S., Fiadeiro, J., Sheth, A., eds.: Business Process Management, volume 4102 of LNCS, Berlin, Springer Verlag (2006) 145–160
19. Aalst, W., Hofstede, A., Weske, M.: Business Process Management: A Survey. In Aalst, W., Hofstede, A., Weske, M., eds.: Business Process Management, volume 2678 of LNCS, Berlin, Springer Verlag (2003) 1–12
20. Sangiorgi, D., Walker, D.: The π-calculus: A Theory of Mobile Processes. Paperback edn. Cambridge University Press, Cambridge (2003)
21. Aalst, W., Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases **14** (2003) 5–51
22. Keller, G., Nüttgens, M., Scheer, A.: Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Technical Report 89, Institut für Wirtschaftsinformatik, Saarbrücken (1992)
23. OMG: UML 2.0 Superstructure Final Adopted specification. (2003)
24. OMG.org: Business Process Modeling Notation. 1.0 edn. (2006)
25. Aalst, W., Hee, K.: Workflow Management. MIT Press (2002)
26. Milner, R.: A Calculus of Communicating Systems. Volume 94 of LNCS. Springer Verlag (1980)
27. Puhlmann, F.: A Tool Chain for Lazy Soundness. In: Demo Session of the 4th International Conference on Business Process Management, CEUR Workshop Proceedings. Volume 203., Vienna (2006) 9–16
28. Basten, T.: In Terms of Nets: System Design with Petri Nets and Process Algebra. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (1998)