Interaction Soundness for Service Orchestrations

Frank Puhlmann and Mathias Weske

Business Process Technology Group
Hasso-Plattner-Institute for IT Systems Engineering
at the University of Potsdam
D-14482 Potsdam, Germany
{puhlmann,weske}@hpi.uni-potsdam.de

Abstract. Traditionally, service orchestrations utilize services according to a choreography where they are a part of. The orchestrations as well as the choreographies describe pre-defined sequences of behavior. This paper investigates if a given orchestration can be enacted without deadlocks, i.e. is interaction sound, inside an environment made up of different services. In contrast to existing approaches, we utilize link passing mobility to directly represent dynamic binding as found in service oriented architectures. Thus, the sequences of interaction behavior are not statically pre-defined but rather depend on the possible behavior of the services in the environment.

1 Introduction

Service oriented architectures (SOA) comprise service orchestrations and choreographies [1]. While orchestrations resemble the internal processes of services, choreographies defines how different services should interact with each other. In this paper we focus on orchestrations that are enacted inside an environment of different services. Instead of reasoning on pre-defined sequences of interactions as given by a choreography, we would rather like to know if a certain orchestrations works seamlessly, i.e. without deadlocks, inside an environment made up of different services as possible interaction partners. The services are not statically connected to the orchestration, but are dynamically bound. For this dynamic binding to take place, we consider a service broker able to return semantically matching services that, however, might have different interaction behaviors. To cope with these different behaviors, we present an approach to determine if a certain orchestration is capable of interacting with a given set of dynamically bound services regardless of a pre-defined behavior.

The approach introduced, denoted as *interaction soundness*, utilizes lazy soundness for deciding deadlock freedom of orchestrations without considering interactions with the environment [2]. Interaction soundness extends lazy soundness by taking these interactions into account. As already motived in [1], the core of a SOA is dynamic discovery and binding of interaction partners. Lazy soundness has been chosen because it can be proven using bisimulation techniques of a process algebra, the π-calculus. The π-calculus in turn supports link passing

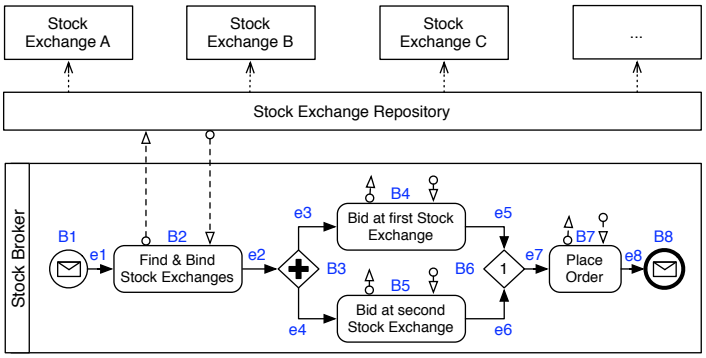

Fig. 1. Stock Exchange Choreography.

mobility, a key feature required for representing dynamic binding [3]. An extended discussion on our motivation for using the π-calculus can be found in [4]. Furthermore, the paper builds on existing work on using the π-calculus for business process management and service oriented architectures [5,6].

The remainder of the paper is structured as follows. We first motivate the topic by introducing an example and refer to related work. Preliminaries are introduced, including π-calculus as well as the representation of orchestrations and choreographies in it. Based thereon, interaction soundness is defined. The paper concludes with a demonstration of existing tool support and a discussion of the achievements.

2 Motivation and Related Work

To motivate the topic, an example is shown in figure 1, denoted in a slightly extended variant of the Business Process Modeling Notation (BPMN). The example describes the orchestration of a *Stock Broker* service and its environment. The stock broker offers the ability of bidding at two different stock exchanges at the same time and place the order at the first stock exchange responding positive, i.e. where the order can be placed. This functionality is realized inside the orchestration using a discriminator pattern [7], denoted as a BPMN gateway containing the number of required incoming edges. Since there are many stock exchanges available with different properties such as fees, rates, and of course business hours, a *Stock Exchange Repository* is contained as a service in the environment. It is invoked as the first task of the stock broker, *Find & Bind Stock Exchanges*. The repository has knowledge about a number of *Stock Exchanges*, connected in the BPMN diagram using associations. Two of them matching the requested conditions are returned to the stock broker. The stock broker is now able to dynamically bind to the stock exchanges formerly unknown. This is denoted using in- and outgoing message flows at the activities *Bid at first Stock*

Exchange and *Bid at second Stock Exchange*. Each stock exchange returns a special token if the bid has been accepted. This token is used inside *Place Order* to place the order at the corresponding stock exchange. Of course, only the successful bidder should be able to place the order.

We now argue why reasoning regarding the soundness of the example is far from trivial. First of all, the stock broker's orchestration contains a discriminator. As already shown in [2], a discriminator leaves running (lazy) activities behind, i.e. the second activity before the discriminator remains active after the first one has completed. This might even be true if the final activity of the orchestration has already been reached. In terms of Petri nets, token remain in the net. According to the Petri net based soundness definition [8], however, an orchestration that contains tokens after the final marking has been reached, is not sound. Secondly, the orchestration is contained inside an environment, where the services are dynamically bound at runtime. Thus, reasoning includes all combinations of services that can be potentially bound.

To overcome these problems, we propose adapting lazy soundness based on the π-calculus for interaction soundness. Lazy soundness takes care of lazy activities, thus solving the first problem (an extended discussion including other kinds of soundness can be found in [2]). More importantly, the π-calculus features link passing mobility, required for describing and reasoning on dynamic binding [3].

Related work comprises for instance Martens using Petri nets [9], Bordeaux and Salan using CCS [10], or Busi et al. using a proprietary process algebra [11]. However, these approaches do not adress dynamic binding. Recent research on interaction patterns by Barros et al. in general [12] and SOA in particular by Guidi and Lucci [13] showed that mobility is indeed required for service oriented architectures.

3 Preliminaries

This section introduces the π-calculus and the representation of orchestrations and environments in it.

3.1 The π-calculus

The π-calculus is an algebra for the formal description and analysis of concurrent, interacting processes with support for link passing mobility. It is based on names and interactions used by processes defined according to [14].

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

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

P and M denote the processes and summations of the calculus. The informal semantics is as follows: $P|P'$ is the concurrent execution of P and P', $\mathbf{v}zP$ is the restriction of the scope of the name z to P, $!P$ is an infinite number of copies of P, and $P(y_1, \cdots, y_n)$ denotes parametric recursion. $\mathbf{0}$ is inaction, a process that can do nothing, $M + M'$ is the exclusive choice between M and M'. The actions of the calculus are given by π. The output prefix $\overline{x}\langle\tilde{y}\rangle.P$ sends a sequence of names $\tilde{y}$ over the co-name $\overline{x}$ and then continues as P. The input prefix $x(\tilde{z})$ receives a sequence of names over the name x and then continues as P with $\tilde{z}$ replaced by the received names (written as $\{^{n\tilde{a}me}/_{\tilde{z}}\}$). Matching input and output prefixes might communicate, leading to an interaction. The unobservable prefix $\tau.P$ expresses an internal action of the process, 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 process identifiers and lower case letters for names. The formal semantics of the π-calculus is based on a transition system. We only give short definitions of the required concepts and refer to [14,15] for details.

Definition 2 (Transition Sequence). *A sequence of interactions on names or unobservable actions is denoted as $P \xrightarrow{\alpha} P'$, where α describes the sequence of actions required to transform a process P to P'.* □

Thus, a transition sequence describes how a certain state P of a process is transfered to another denoted P'.

Definition 3 (Bound and Free Names). *The π-calculus has two operators for name binding, $x(z)$ and $\mathbf{v}zP$. In both cases the name z is* bound *inside process P. Names which are not bound by a name binding operator are called* free names *of a process.* □

Bound names can not be accessed from processes outside of P. Free names can be used for interactions between different processes. The free names of a processes can furthermore be observed outside of the process P for reasoning on bisimulation equivalence.

Definition 4 (Weak Open Bisimulation Equivalence). *Informally, two π-calculus processes P and Q are* weak open bisimulation equivalent*, denoted as $P \approx^o Q$, if they have the same observable behavior regarding their free names while abstracting from all internal actions.* □

Weak open bisimulation is used to prove interaction soundness later on. Formal details can be found in [15].

3.2 Orchestrations in the π-calculus

Orchestrations can be formalized using set theory and π-calculus. The former is used to denote the static structure of the orchestration called *process graph*; the latter gives a formal semantics to a process graph. Since orchestrations are usually denoted graphically, a breakdown from graphical representations over process graphs up to π-calculus is possible.

Definition 5 (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 *is a finite, non-empty set of nodes.*
- $E \subseteq (N \times N)$ *is a set of directed edges.*
- $T : N \rightarrow 2^{TYPE}$ *is a function mapping nodes to sets of types.*
- $A : N \nrightarrow KEY \times VALUE$ *is a partial function mapping nodes to key/value pairs.* □

The nodes N of a process graph define the activities (incl. routing elements) of an orchestration, and the directed edges E define dependencies between activities. Each node can have none, one, or more types assigned by the function T. Furthermore, each node can hold optional attributes represented by key/values pairs assigned by the function A. Details will become more clear by looking at the process graph of the stock broker from figure 1. We now utilize the identifiers from the figure:

Example 1 (Process Graph of the Stock Broker Orchestration).

1. $N = \{B1, B2, B3, B4, B5, B6, B7, B8\}$
2. $E = \{(B1, B2), (B2, B3), (B3, B4), (B3, B5), (B4, B6), (B5, B6), (B6, B7), (B7, B8)\}$
3. $T = \{(B1, \{\text{Start Event}\}), (B2, \{\text{Task}\}), (B3, \{\text{AND Gateway}\}), (B4, \{\text{Task}\}), (B5, \{\text{Task}\}), (B6, \{\text{N-out-of-M-Join}\}), (B7, \{\text{Task}\}), (B8, \{\text{End Event}\})\}$
4. $A = \{(B6, (\text{continue}, 1))\}$ □

Each node of the orchestration is denoted as an element of N, whereas sequence flows between nodes are denoted in E. The types are simply represented as the textual name of the corresponding BPMN element in T. Other notations like *EPCs* or *UML2 Activity Diagrams* cause other types. The discriminator is denoted as a special kind of *n–out–of–m–join* with $n = 1$. This threshold is denoted in the set A.

However, since a process graph only denotes a static structure of an orchestration, in particular even with different types for the nodes, a formal semantics is given by mapping the process graph to π-calculus expressions. Therefore, we assume each node of the process graph to represent one of the workflow patterns [7]. The steps for mapping process graphs to π-calculus can then be sketched as follows (see [2] for a complete description).

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

1. Assign all nodes of P a unique π-calculus process identifier $N1 \cdots N|P_N|$.
2. Assign all edges of P a unique π-calculus name $e1 \cdots e|P_E|$.
3. Define the π-calculus processes according to the π-calculus mapping of the workflow patterns found in [5,3] as given by the type of the corresponding node. Each functional part of an activity is represented by the unobservable prefix τ since it is abstracted from concrete realizations.
4. Define a global process $N = (\mathbf{v}e1, \cdots, e|P_E|) \prod_{i=1}^{|P_N|} Ni$. □

Section 5 contains a π-calculus mapping of the example.

3.3 Environments in the π-calculus

Environments can be split into static ones with pre-defined bindings and such supporting dynamic binding. For static environments, a corresponding concept to a process graph, called *interaction graph IG*, can be introduced. An interaction graph relates several process graphs by *interaction flow*, according to *Message Flow* in BPMN.

However, the focus of this paper is on environments that support dynamic binding as given in the example. These environments are closely linked to the service interaction patterns by Barros et al. [12]. The patterns describe possible interaction behavior between services. To our knowledge, there exists no graphical notation that supports the representation of dynamic binding. Since a graphical representation is missing, we define the environments from scratch in π-calculus. This is done according to [3], where correlations and dynamic service invocation in π-calculus have been introduced. A synchronous service invocation is denoted in the π-calculus as:

$$A = \overline{b}\langle msg \rangle.A'$$
$$B = b(msg).B' \ ,$$

where A is the service requester and B is the service provider. The formalization leaves it open if A knows the link b at design time or acquired it during runtime. If the system is defined as

$$S = (\mathbf{v}b)(A \mid B) \ ,$$

A and B share the link b since design time. Using link passing mobility in π-calculus, we can model a repository $R = \overline{lookup}\langle b \rangle.R$ that transmits the link at runtime:

$$S = (\mathbf{v}lookup)(lookup(b).A \mid (\mathbf{v}b)(B \mid R)) \ .$$

An overview of formalizing more complex service interaction patterns in π-calculus including further references is given in [6].

4 Interaction Soundness

This section derives interaction soundness for orchestrations based on lazy soundness. Lazy soundness proves an orchestration containing lazy activities to be free of deadlocks and livelocks. Interaction soundness extends lazy soundness by incorporating the interactions between the orchestration and the environment.

4.1 Lazy Soundness

Lazy soundness requires *structural soundness* and *semantic reachability*. A process graph representing an orchestration is called structurally sound if it has exactly one initial node, exactly one final node and all nodes lie on a path from the initial to the final node.

Definition 6 (Structural Sound). *A process graph $P = (N, E, T, A)$ is structural sound if and only if:*

1. *There is exactly one initial node $N_i \in N$.*
2. *There is exactly one final node $N_o \in N$.*
3. *Every node is on a path from N_i to N_o.* □

Semantic reachability extends reachability by taking the semantics of the nodes into account.

Definition 7 (Semantic Reachability). *A node $N_1 \in N$ of a process graph $P = (N, E, T, A)$ is semantically reachable from another node $N_2 \in N$, denoted as $N_1 \rightsquigarrow N_2$, if and only if there exists a path leading from N_1 to N_2 according to the semantics of all nodes.*

Regarding the mapping of a π-calculus process from a process graph, a π-calculus process P_1 representing a node is semantically reachable from another π-calculus process P_2 representing a node, if and only if there exists a transition sequence from the functional abstraction τ of P_1 to the functional abstraction τ of process P_2. Lazy soundness is now defined as follows:

Definition 8 (Lazy Sound). *A structural sound process graph $P = (N, E, T, A)$ is lazy sound if and only if:*

1. *The final node N_o must be semantically reachable from every node $n \in N$ semantically reachable from the initial node until N_o has been reached for the first time. Formally: $\forall n \in N$ with $N_i \rightsquigarrow n : n \rightsquigarrow N_o$ holds until N_o has been reached for the first time.*
2. *The final node N_o is reached exactly once.*

Definition 8 states that a lazy sound process graph representing a business process is deadlock and livelock free as long as the final node has not been executed (8.1). Once the final node has been executed, other nodes might still be executed, however they do not semantically reach the final node again (8.2). Lazy soundness can be proven by tracing the initial and the final activity of a process graph mapped to π-calculus processes.

4.2 Interaction Soundness for Service Orchestrations

Interaction soundness is defined for service graphs that enhance a process graph with interaction behavior.

Definition 9 (Service Graph). *A service graph extends a process graph by adding in- or outbound interaction edges used as a behavioral interface. Formally, $SG = (PS, C, L)$:*

- *$PS = (N_{PS}, E_{PS}, T_{PS}, A_{PS})$ is a structural sound process graph.*
- *$C \subseteq (N_{PS} \times \perp) \times (\perp \times N_{PS})$ is a set of directed interaction edges.*
- *$L \subseteq (C \times LABEL)$ is a set of labels of directed interaction edges.* □

PS is an orchestration describing the internal process of a service. Interactions with the environment are denoted by C, representing in- and outgoing communication (e.g. Message Flows in BPMN). The symbol $\perp$ is used as a connector to the environment. L attaches labels based on π-calculus names used to denote channels and data (examples can be found in section 5).

A counterpart to a service graph is given by an environment that can be utilized by the service graph.

Definition 10 (Environment). *Let SG be a service graph. An environment E for SG is given if E utilizes at least one in- or outgoing interaction edge C of SG by providing a matching process structure.*

Furthermore, a service graph SG unified with an environment E is denoted as $SG \uplus E$. Interaction soundness is now given by:

Definition 11 (Interaction Soundness). *A service graph SG is interaction sound regarding environment E if and only if $SG \uplus E$ is lazy sound.*

Interaction soundness states that a service graph representing an orchestration is deadlock and livelock free under consideration of all related interactions with the environment as long as the final activity of the orchestration has not been reached.

4.3 Reasoning on Interaction Soundness

Since interaction soundness is defined using service graphs and environments that do not yet have a formal semantics like process graphs, we now show how to enhance them for reasoning. First of all, the π-calculus mapping of the process graph contained in the service graph is annotated with π-calculus names used for interaction with the environment. This is done according to the labels and directed interaction edges of the service graph. Secondly, the environment is defined using π-calculus processes being able to interact with the π-calculus mapping of the service graph according to [3,6]. This is currently a manual task.

Once the π-calculus representation of a system consisting of the π-calculus mapping of a service graph and an environment has been defined, it can be enhanced for reasoning on lazy soundness as described in [2]. Basically the π-calculus process representing the initial activity of the orchestration is enhanced with the free name i and the π-calculus process representing the final activity is enhanced with the free name $\overline{o}$. Due to that, we are able to observe the occurrence of the initial activity and the final activity.

The distinction between interaction soundness and lazy soundness is given by the fact that not only the structure of the service graph is checked for conformance, but also the agile interaction with the environment using link passing mobility. For the π-calculus mapping of an orchestration and an environment to be interaction sound, i and $\overline{o}$ have to be observed exactly once for all possible transition sequences, including the ones between the participants. Thereby, i is just a helper to denote the start of the orchestration. The interesting part is

$\overline{o}$. If $\overline{o}$ is not observed for all possible transition sequences, the orchestration contains a deadlock or livelock since property (1) of definition 8 is violated. If $\overline{o}$ is observed more than once, property (2) of definition 8 is violated, i.e. the orchestration contains uncontrolled loop or parallel process structures. To prove the π-calculus representation of an orchestrations and an environment to be interaction sound, it is compared for weak open bisimulation against a manually proved lazy sound π-calculus process given by $S_{LAZY} = i.\tau.\overline{o}.\mathbf{0}$.

Proposition 1. *A π-calculus representation SYS of a service choreography consisting of (1) a π-calculus mapping of an orchestration annotated with the free names i for the π-calculus process representing the initial and $\overline{o}$ for the π-calculus process representing the final activity of the orchestration, and (2) a π-calculus representation of a corresponding environment is interaction sound if and only if $SYS \approx^o S_{LAZY}$.*

5 Example and Tool Support

This section discusses how the theoretical results described in the last section can be applied using existing π-calculus tools like the Mobility Workbench (MWB) [16]. We utilize the example shown in figure 1.

The π-calculus representation of the stock broker's orchestration is generated from the BPMN diagram using a tool chain developed at our group.[1] The repository and different stock exchanges have been modeled manually, since their agile interaction behavior can not be modeled in BPMN. Interaction between the different participants is represented using π-calculus names. The π-calculus processes corresponding to figure 1 are printed below. To enable direct reasoning, the notation of the Mobility Workbench has been used, i.e. output prefixes are written as $'x$ instead of $\overline{x}$ and ^ denotes **v**. Processes in the π-calculus are denoted as `agents`.

Example 2. Pi-Calculus Processes for the Stock Exchange Choreography.

```
agent SE_A(ch) = (^o)ch(b).t.'b<o>.o.SE_A(ch)
agent SE_B(ch) = (^o)ch(b).t.'b<o>.o.SE_B(ch)
agent SE_C(ch) = (^o)ch(b).t.'b<o>.o.SE_C(ch)

agent R(r,s1,s2,s3)=r(ch).'ch<s1>.r(ch).'ch<s2>.R(r,s1,s2,s3) +
      r(ch).'ch<s2>.r(ch).'ch<s3>.R(r,s1,s2,s3) + r(ch).'ch<s1>.r(ch).'ch<s3>.R(r,s1,s2,s3)

agent B(i,o,r)=(^e1,e2,e3,e4,e5,e6,e7,e8)( B1(e1,i) | B2(e1,e2,r) | B3(e2,e3,e4) | B4(e3,e5) |
      B5(e4,e6) | B6(e5,e6,e7) | B7(e7,e8) |B8(e8,o))
agent B1(e1,i)=i.t.'e1.0
agent B2(e1,e2,r)=(^ch)e1.'r<ch>.ch(s1).'r<ch>.ch(s2).t.('e2<s1,s2>.0 | B2(e1,e2,r))
agent B3(e2,e3,e4)=e2(s1,s2).t.('e3<s1>.0 | 'e4<s2>.0 | B3(e2,e3,e4))
agent B4(e3,e5)=(^b)e3(s).'s<b>.b(o).t.('e5<o>.0 | B4(e3,e5))
agent B5(e4,e6)=(^b)e4(s).'s<b>.b(o).t.('e6<o>.0 | B5(e4,e6))
agent B6(e5,e6,e7)=(^h,run)(B6_1(e5,e6,e7,h,run) | B6_2(e5,e6,e7,h,run))
agent B6_1(e5,e6,e7,h,run)=e5(o).'h<o>.0 | e6(o).'h<o>.0
agent B6_2(e5,e6,e7,h,run)=h(o).'run<o>.h(o).B6(e5,e6,e7) | run(o).t.'e7<o>.0
agent B7(e7,e8)=e7(o).'o.t.('e8.0 | B7(e7,e8))
agent B8(e8,o)=e8.t.'o.B8(e8,o)
```

[1] `http://bpt.hpi.uni-potsdam.de/twiki/bin/view/Piworkflow/Reasoner`

```
agent SYS(i,o) = (^s1,s2,s3,r)( SE_A(s1) | SE_B(s2) | SE_C(s3) | R(r,s1,s2,s3) | B(i,o,r))

agent S_LAZY(i,o)=i.t.'o.0
```

The first three lines of the example denote simple kinds of services that are used for reasoning. They simply create an order token o, wait for a connection via $ch(b)$, where b is a response channel used to signal back the o token. In between, however, complex computation takes place that is abstracted from by τ. Note that the different services do not differ in their interaction behavior right now, thus we could utilize each of them inside our orchestration.

The process R denotes a very simple kind of a repository that simply returns two arbitrary services. We omitted a complex structure based on lists that would allow arbitrary services to register and to de-register. The stock broker's orchestration is represented in the third block. B is a π-calculus process containing all activities of the stock broker, that in turn are represented according to figure 1 by $B1 \ldots B8$. Note the processes $B2$, where the stock exchanges are found at the repository (`'r<ch>.ch(s1).'r<ch>.ch(s2)`), $B4$ and $B5$, where the stock exchanges are dynamically bound and invoked (`'s<b>.b(o)`). Furthermore, the successful bidding activity forwards the order token to $B7$, where the order is finally placed. To allow activities of the orchestration to be observed according to lazy soundness, $B1$ and $B8$ are enhanced with `i` and `'o` accordingly. The process $SYS(i, o)$ places all participants into a system leaving only i and o as free names. SYS can then be compared to S_LAZY required for deciding lazy soundness.

The formalization given allows reasoning on the orchestration and the environment. The problems motivated in section 2 are solved using the π-calculus representation. Lazy (left–behind) activities before the discriminator are handled using lazy soundness. The π-calculus mapping of the discriminator, found in process $B6$, enables the outgoing sequence flow (denoted using $e7$) exactly once for the first incoming sequence flow (here $e5$ or $e6$). The second incoming sequence flow is simply captured. Furthermore, dynamic binding of different services is represented using link passing mobility as shown in the example. A tool session using MWB to prove interaction soundness is shown below:

```
MWB>weq SYS(i,o) S_LAZY(i,o)
The two agents are equal.
```

The orchestration of the stock broker inside the environment represented by SYS is weak open bisimulation equivalent to S_LAZY, hence the orchestration is interaction sound. Since the orchestration includes the interactions with the repository and stock exchanges, all possible behaviors of the services are acceptable and will not lead to a deadlock. But what happens if one of the possible interaction partners, e.g. one of the services, shows a different interaction behavior? This can be investigated for instance by changing the definition of $SE\_A(ch)$ to wait for a confirmation of the bidding via b before proceeding.

```
MWB>agent SE_A(ch) = (^o)ch(b).b(confirm).t.'b<o>.o.SE_A(ch)
MWB>weq SYS(i,o) S_LAZY(i,o)
The two agents are equal.
```

Once again, the orchestration is interaction sound. This is even true if the "defective" service represented by agent $SE\_A(ch)$ is dynamically bound to our orchestration. In this case, always the second service will be utilized (due to the discriminator). Hence, the orchestration will not deadlock even if a non–matching, defective service is contained in the environment. However, if we introduce a second defective service by changing $SE\_B(ch)$, the possibility of selecting and binding to two defective services exists, thus leading to a serious problem:

```
MWB>agent SE_B(ch) = (^o)ch(b).b(confirm).t.'b<o>.o.SE_B(ch)
MWB>weq SYS(i,o) S_LAZY(i,o)
The two agents are NOT equal.
```

The orchestration contained in the modified system is not interaction sound anymore, since there exist possible combinations of services in it that will lead to deadlock situations.

6 Conclusion

In this paper it has been shown how orchestrations that dynamically bind to services in a given environment can be proved to be interaction sound. The approach presented is generic in a sense that it is not limited to certain kinds of orchestrations or interactions. This is due to the possibility of representing all routing and interaction patterns, either workflow or service interaction ones, in a precise way in π-calculus [5,6]. Thus, existing orchestrations can be formalized and analyzed. The soundness criterion used for interaction soundness, lazy soundness, furthermore supports lazy activities. Since orchestrations can contain lazy activities, these do not disturb the reasoning.

The approach presented in this paper is a starting point for investigating formal properties of dynamic bindings. First of all, existing graphical notations like BPMN do not support the representation of systems containing dynamic binding. Without a graphical representation, user acceptance is limited. So one direction of further work is creating such a notation. Second, tool support for π-calculus is currently limited. The existing tools are not optimized for reasoning on service orchestrations. For instance, they use depth-first search strategies that cause problems regarding the detection of certain loop constructs.

References

1. Burbeck, S.: The Tao of E-Business Services. Available at: `http://www-128.ibm.com/developerworks/library/ws-tao/` (2000)
2. Puhlmann, F., Weske, M.: Investigations on Soundness Regarding Lazy Activities. In Dustdar, S., Fiadeiro, J., Sheth, A., eds.: Proceedings of the 4th International Conference on Business Process Management (BPM 2006), volume 4102 of LNCS, Berlin, Springer Verlag (2006) 145–160
3. Overdick, H., Puhlmann, F., Weske, M.: Towards a Formal Model for Agile Service Discovery and Integration. In Verma, K., Sheth, A., Zaremba, M., Bussler, C., eds.: Proceedings of the International Workshop on Dynamic Web Processes (DWP 2005). IBM technical report RC23822, Amsterdam (2005)

4. 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
5. Puhlmann, F., Weske, M.: Using the Pi-Calculus for Formalizing Workflow Patterns. In van der Aalst, W., Benatallah, B., Casati, F., eds.: Proceedings of the 3rd International Conference on Business Process Management, volume 3649 of LNCS, Berlin, Springer-Verlag (2005) 153–168
6. Decker, G., Puhlmann, F., Weske, M.: Formalizing Service Interactions. In Dustdar, S., Fiadeiro, J., Sheth, A., eds.: Proceedings of the 4th International Conference on Business Process Management (BPM 2006), volume 4102 of LNCS, Berlin, Springer Verlag (2006) 414–419
7. Aalst, W., Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases **14** (2003) 5–51
8. Aalst, W.: Verification of Workflow Nets. In Azéma, P., Balbo, G., eds.: Application and Theory of Petri Nets, volume 1248 of LNCS, Berlin, Springer-Verlag (1997) 407–426
9. Martens, A.: Analyzing Web Service based Business Processes. In Cerioli, M., ed.: Proceedings of Intl. Conference on Fundamental Approaches to Software Engineering (FASE'05). Volume 3442 of Lecture Notes in Computer Science., Springer-Verlag (2005)
10. Bordeaux, L., Salaün, G.: Using Process Algebra for Web Services: Early Results and Perspectives. In Shan, M.C., Dayal, U., Hsu, M., eds.: TES 2004, volume 3324 of LNCS, Berlin, Springer-Verlag (2005) 54–68
11. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and Orchestration: A Synergic Approach to System Design. In Benatallah, B., Casati, F., Traverso, P., eds.: Proceedings of the 3rd International Conference on Service-oriented Computing, volume 3826 of LNCS, Berlin, Springer-Verlag (2005) 228–240
12. Barros, A., Dumas, M., ter Hofstede, A.: Service Interaction Patterns. In van der Aalst, W., Benatallah, B., Casati, F., eds.: Proceedings of the 3rd International Conference on Business Process Management, volume 3649 of LNCS, Berlin, Springer-Verlag (2005) 302–318
13. Guidi, C., Lucchi, R.: Mobility mechanisms in Service Oriented Computing. In: Proc. of 8th International Conference on on Formal Methods for Open Object-Based Distributed Systems (FMOODS06). (2006 (to appear))
14. Sangiorgi, D., Walker, D.: The π-calculus: A Theory of Mobile Processes. Paperback edn. Cambridge University Press, Cambridge (2003)
15. Sangiorgi, D.: A Theory of Bisimulation for the Pi-Calculus. In: CONCUR '93: Proceedings of the 4th International Conference on Concurrency Theory, Berlin, Springer-Verlag (1993) 127–142
16. Victor, B., Moller, F., Dam, M., Eriksson, L.H.: The Mobility Workbench. Available at: `http://www.it.uu.se/research/group/mobility/mwb` (2005)